

Angelo Gallippi

Microsoft QBASIC

Corso completo

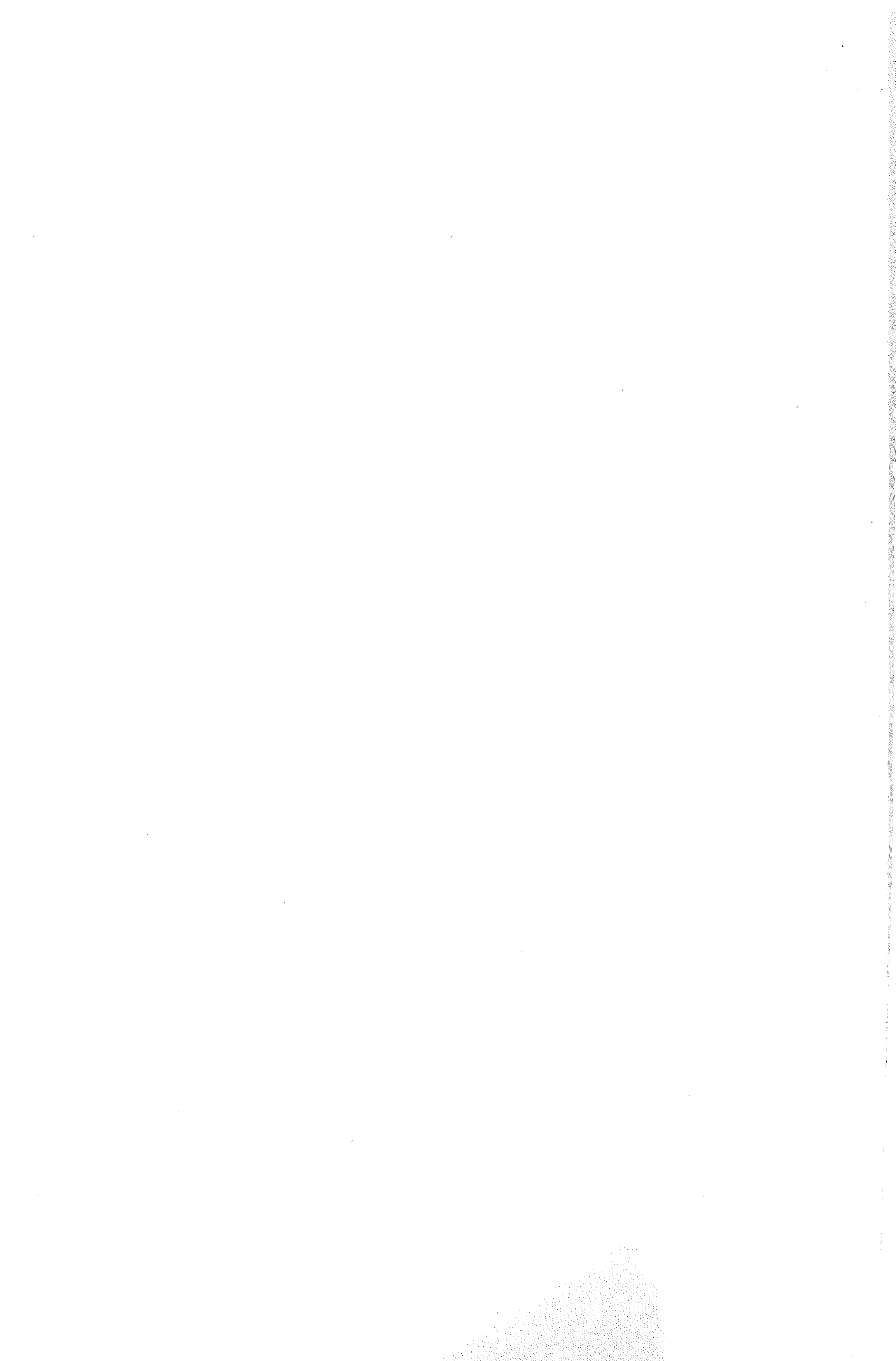


tecniche nuove



Angelo Gallippi

Microsoft QBASIC Corso completo



Angelo Gallippi

Microsoft QBASIC Corso completo

tecniche nuove

tel. (02) 75701, telefax (02) 7610351

ISBN 88 481 00198

L'Editore dichiara la propria disponibilità a regolarizzare eventuali omissioni o errori di attribuzione.

Stampa: Grafiche D.L. Melzo (MI)
Finito di stampare nel mese di settembre 1994
Printed in Italy

Indice

Capitolo 1
Introduzione XI
Perchè la programmazione? 1
Perchè il QBASIC 1

Capitolo 2
Primi elementi 3
Avvio di QBASIC 5
La videata di QBASIC 6
La barra dei menu 7
Perchè due finestre? 8
La finestra "Senza titolo" 8
La finestra "Immediato" 9
L'ultima riga 9

Capitolo 3
Numeri e lettere 11
Tipi di dati numerici 13
Notazione scientifica e floating point 13
Tabella dei codici ASCII 14

Capitolo 4
Variabili e costanti 17
Che cosa sono le variabili e le costanti 19
Assegnazione di valori da programma (LET, CONST) 19
Tipi di variabili 20
Dichiarazione dei tipo di variabili (DEF, CLEAR) 21

Capitolo 5
Visualizzazione di testo sullo schermo 23
Righe, colonne e celle 25
Istruzione PRINT 26
Visualizzazione di dati formattati (PRINT USING) 27
Codice di controllo per espressioni di stringa 28
Codici di controllo per espressioni numeriche 29

Salto di spazi (SPC, SPACE$) 30
Avanzamento a una colonna specifica (TAB) 31
Posizionamento del cursore (LOCALE, CSRLIN, POS) 31
Cambiamento del numero di colonne o di righe (WIDTH) 32
Stampa su carta (LPRINT, LPRINT USING) 32
Finestra di testo (VIEW PRINT, CLS 2) 33

Capitolo 6
Funzioni ed espressioni numeriche **35**
Che cos'è una funzione 37
Funzioni numeriche 37
Espressioni numeriche 41

Capitolo 7
Stringhe di caratteri **43**
Che cos'è una stringa 45
Dichiarazione di costanti e variabili (CONST, DEFSTR, AS STRING) 45
Stringhe di lunghezza variabile e fissa (DIM, RSET) 46
Concatenazione di stringhe e di caratteri (STRING$) 47
Confronto di stringhe 48
Funzioni di stringa 50
Funzioni varie 54
Immissione di dati dalla tastiera 55
Il programma: primi concetti 58
Commenti in un programma (REM) 59
Fine di un programma (END) 59

Capitolo 8
Strutture cicliche **61**
Che cos'è un ciclo 63
Ciclo FOR...NEXT 63
Annidamento di cicli 65
Uscita da un ciclo FOR...NEXT con EXIT FOR 66
Pausa in un programma con FOR...NEXT 67
Espressioni booleane 67
Operatori booleani 67
Ciclo WHILE...WEND 70
Cicli DO...LOOP 71
Confronto tra DO...LOOP e WHILE...WEND 76
Uscita da un ciclo DO...LOOP con EXIT DO 76
Menu di scelta con DO...LOOP 77

Capitolo 9
Strutture di decisione **79**
Programmi flessibili 81
IF...THEN...ELSE su una sola linea 81
IF...THEN...ELSE nella forma a blocchi 82
Istruzione SELECT CASE 84
Confronto tra SELECT CASE e IF...THEN...ELSE 87

Capitolo 10
Vettori e matrici **89**
Che cosa sono i vettori e le matrici 91
Dimensionamento (DIM, OPTION BASE) 92
Opzione dell'istruzione DIM 93
Cancellazione di un vettore (ERASE) 93
Assegnazione di valori 93
Dimensionamento dinamico 94
Interpolazione di ordine n 95
Lettura e assegnazione di valori (READ...DATA) 98
Rilettura di costanti (RESTORE) 99
Tecniche di lettura dei dati 100
Ricerca dell'elemento massimo di un vettore 101
Ordinamento delle componenti di un vettore (SWAP) 102
Altre operazioni con i vettori 104

Capitolo 11
Programmazione modulare **105**
Che cosa sono i moduli di programma 107
Subroutine (ON...GOSUB) 108
Confronto fra ON...GOSUB e SELECT CASE 109
Procedure 110
Sottoprogrammi (CALL, SUB...END SUB) 111
Editazione di un sottoprogramma 113
Funzioni (FUNCTION...END FUNCTION) 114
Confronto tra FUNCTION e DEF FN 115
Argomenti e parametri delle procedure 116
Passaggio di costanti ed espressioni 117
Passaggio di argomenti per riferimento 118
Passaggio di argomenti per valore 119
Variabili statiche e automatiche (STATIC) 120
Ambito di una variabile (DIM SHARED) 120
Pausa in un programma con DO...LOOP 122
Procedure ricorsive 123

Capitolo 12
File di programma e di dati **125**
File di programma: salvataggio 127
File di programma: apertura 127
File di dati 127
File sequenziali e ad accesso casuale 128
Salvataggio e apertura (OPEN) 129
Chiusura (CLOSE, RUN) 129
File sequenziali: creazione (WRITE #) 130
Apertura per aggiunta (APPEND) e per scrittura (OUTPUT) 131
File sequenziali: lettura (INPUT #, EOF) 132
Altre possibilità di lettura 133
Istruzione LINE INPUT # 133

Funzione INPUT$ 133

Capitolo 13
File ad accesso casuale **135**
Organizzazione dei record 137
Aggiunta di dati 137
Definizione dei campi del record (TYPE...END TYPE) 137
Apertura del file e indicazione della lunghezza del record (LEN=) 138
Immissione dei dati nel record e sua memorizzazione (PUT#) 138
Aggiunta di dati nelle precedenti versioni di BASIC (FIELD, MK...) 140
Lettura globale di dati 141
Lettura di dati nelle precedenti versioni di BASIC (CV...) 141
Richiamo di record tramite i numeri di record 142
Lettura/scrittura binaria di file 143
Altre operazioni con file (NAME, KILL, CHDIR, MKDIR, FILES) 144

Capitolo 14
La grafica **145**
Schede grafiche 147
Risoluzione e pixel 147
Istruzione SCREEN 148
Disegno di singoli pixel (PSET, PRESET) 150
Disegno di segmenti (LINE) 151
Coordinate relative (STEP) 151
Disegno di rettangoli (B, BF) 153
Linee tratteggiate (STILE) 153
Disegno di cerchi ed ellissi (CIRCLE) 156
Disegno di ellissi (aspetto) 156
Disegno di archi 157
Disegno di una torta 159
Definizione di una finestra grafica (CLS 1, VIEW) 160
Ridefinizione delle coordinate della finestra (WINDOW) 161
Uso del colore 162
Colore di primo piano e di sfondo (COLOR) 162
Cambiamento del colore di primo piano o di sfondo
(PALETTE e PALETTE USING) 164
Riempimento di figure (PAINT) 165
Riempimento con un colore 166
Riempimento con un motivo monocromatico 167
Riempimento con un motivo a colori 169
Linguaggio di macro grafiche (DRAW) 171
Stampa di grafici (GRAPHICS) 173

Capitolo 15
Il suono **175**
Istruzione BEEP 177
Istruzione SOUND 177
Linguaggio di programmazione musicale (PLAY) 178
Comandi di ottava 179

Comandi di durata 180
Comandi di tempo 181
Comandi di operazione 181

Capitolo 16
Intercettazione degli errori **183**
Prevedere gli errori 185
Attivazione dell'intercettazione degli errori (ON ERROR GOTO) 185
Codici di errore (ERR) 185
Routine di gestione degli errori 186
Uscita dalla routine di gestione degli errori (RESUME [NEXT]) 188

Indice analitico 191

PRESENTAZIONE

La Microsoft, società produttrice e distributrice di MS-DOS - il più diffuso sistema operativo per personal computer - ha dotato questo prodotto, fin dalle sue prime versioni, di un linguaggio di programmazione, l'altrettanto famoso BASIC, che ha permesso a milioni di utenti non professionali di avvicinarsi per la prima volta a questa impegnativa attività informatica che è la programmazione.
È quindi naturale che anche l'ambiente operativo Windows, che in un certo senso raccoglie l'eredità del DOS e facilita l'uso del computer a platee ancora più vaste di utenti, fosse dotato di una propria versione di BASIC, denominata QBASIC. Se la nuova versione del linguaggio si presenta più "amichevole" nella sua interazione con l'utente rispetto alle precedenti, essa presuppone tuttavia, come ogni linguaggio di programmazione, che l'utente abbia ben chiari i principi e le tecniche fondamentali della programmazione, prima ancora di poter trarre vantaggio dagli aiuti in linea e dalla sintassi delle singole istruzioni richiamabili comodamente a video.
Questo manuale, che riprende e aggiorna le parti principali del precedente volume "Corso di BASIC" dello stesso autore, intende fornire uno strumento di lavoro e una serie di suggerimenti utili a quanti intendano imparare l'affascinante tecnica della programmazione, e assumere un controllo "completo" delle operazioni da far compiere al proprio computer, al di là cioè del normale utilizzo quotidiano della maggior parte degli operatori. A questa categoria di utenti il volume permetterà di passare in maniera semplice - e forse anche divertente - dalla fase di "utilizzatori di computer" a quella di "informatici".

Capitolo 1

Introduzione



PERCHÉ LA PROGRAMMAZIONE?

I termini "programma" e "programmazione" sono abbastanza diffusi anche al di fuori degli specialisti informatici: ad esempio, si può scegliere il programma da far eseguire a una lavatrice o una lavastoviglie, mentre si può programmare un videoregistratore perché registri una o più trasmissioni televisive. In tutti questi casi, la programmazione determina l'esecuzione di un certo numero di operazioni in maniera sequenziale (cioè l'una dopo l'altra), in genere in un tempo futuro, e senza la necessità di ulteriori interventi umani. I vantaggi sono diversi: l'operatore può eseguire un'altra attività mentre la macchina lavora, è sicuro che l'azione programmata avvenga nei tempi e modi voluti e, soprattutto, può ottenere tutto ciò con la semplice pressione di pochi tasti.

Nel caso dei computer - che si possono considerare delle macchine enormemente più complesse degli elettrodomestici - la programmazione permette certo di fare tutto ciò, ma anche di più; essa infatti:

- **permette di far eseguire al computer delle operazioni che si susseguono non solo in maniera prefissata, ma anche in un ordine che dipende dal risultato delle operazioni stesse;**
- **permette di far svolgere al computer un numero di compiti assai superiore a quelli previsti dal costruttore: anzi, questi si pone solo marginalmente il problema di "che cosa" può fare il suo computer, mentre opera sostanzialmente per aumentarne l'efficienza e ridurne l'ingombro.**

Spetterà poi al programmatore il compito di istruire il computer a eseguire un compito piuttosto che un altro, in un modo facile per l'utente oppure difficile, e con un insieme ridotto oppure ridondante di istruzioni.

PERCHÉ IL QBASIC?

La programmazione è quindi *necessaria* per il funzionamento di qualsiasi computer, tanto è vero che ciascuno di essi viene venduto con un corredo più o meno ampio di programmi o *software* già presenti al suo interno. Tali programmi sono tradizionalmente suddivisi nelle due categorie del *software di base* e del *software applicativo,* che svolgono rispettivamente operazioni *orientate alla macchina* e *orientate all'utente.*
Un'operazione orientata alla macchina potrebbe essere, per esempio, la predisposizione di un dischetto magnetico per la successiva registrazione dei dati su di esso *(formattazione),* o la visualizzazione sullo schermo dei titoli di tutte le registrazioni *(file)* presenti su un disco magnetico.
Un'operazione orientata all'utente potrebbe essere invece la possibilità di scrivere una serie di caratteri sullo schermo e di trasformarli in un testo scritto su carta tramite la pressione di pochi tasti, o la possibilità di eseguire operazioni matematiche come con una calcolatrice.
Il software applicativo oggi in commercio copre la maggior parte delle necessità di un utente di computer: scrittura, archiviazione, contabilità, addestramento, grafica, suono. Esso tuttavia presenta anche delle limitazioni:

- è poco o affatto **modificabile**: l'utente può infatti ottenere tutte e sole le funzionalità

previste dal progettista, ma raramente può alterare il programma per adattarlo a un'esigenza non prevista in fase di acquisto, o modificatasi nel periodo di utilizzo;
- copre applicazioni **standard** - cioè quelle per le quali ci sia un mercato sufficientemente ampio da giustificare gli elevati investimenti economici necessari per lo sviluppo di un pacchetto applicativo - ma si disinteressa a quelle che possono essere le esigenze di pochi o di un utente singolo;
- attua necessariamente un **compromesso** tra esigenze contrastanti, quali velocità di elaborazione e occupazione di memoria da una parte, facilità di utilizzo e gradevolezza dell'interfaccia grafica dall'altra.

Chi desideri superare una o più di queste limitazioni deve necessariamente affidarsi all'opera di un programmatore oppure, ed è la cosa più appassionante, diventare programmatore egli stesso. In questo secondo caso un buon punto di partenza è lo studio del linguaggio QBASIC, che presenta diversi vantaggi:

- è fornito insieme all'ambiente operativo Windows, quindi è completamente **gratuito** per la maggior parte di utenti di personal computer;
- è molto **potente**, supportando gran parte delle caratteristiche dei linguaggi più avanzati (programmazione modulare, ricorsività, costruzione DO CASE,...) pur accettando costrutti più vecchi dalle precedenti versioni di BASIC;
- permette di scrivere **programmi** di diversa complessità: da molto brevi per risolvere compiti semplici (ad esempio, trovare tutti gli anagrammi di una parola, eseguire un'interpolazione di ordine qualsiasi, ordinare alfabeticamente delle parole), a molto lunghi, per risolvere praticamente ogni tipo di problema suscettibile di essere affrontato con un personal computer.

Capitolo 2

Primi elementi

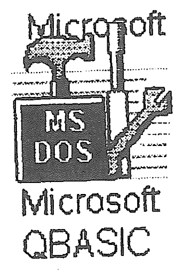


AVVIO DI QBASIC

Dal punto di vista tecnico, QBASIC è un programma o meglio un file di tipo eseguibile (il suo nome è infatti QBASIC.EXE), cioè codificato in un formato comprensibile dal calcolatore ma non leggibile dall'utente. Esso è di solito memorizzato in una zona del disco rigido *(sottoindirizzario)* chiamata DOS o MSDOS.
QBASIC può essere avviato sia se sullo schermo è presente il prompt del DOS

```
C:\>
```

(cioè se si lavora "in ambiente DOS"), sia se è visualizzata l'icona



di QBASIC nella finestra delle Applicazioni di Windows 3.1 (cioè se si lavora "in ambiente Windows"). Nel primo caso si dovrà scrivere

```
cd dos\qbasic
```

Nel secondo caso sarà sufficiente posizionare il puntatore del mouse sull'icona di QBASIC ed effettuare un doppio click sul pulsante attivo del mouse (in genere quello di sinistra). In ambiente DOS il comando qbasic può essere seguito da una o più "opzioni" (ad esempio, qbasic/B), che producono gli effetti indicati in Tabella 1.

OPZIONE	EFFETTO
\B	Permette di utilizzare un monitor monocromatico composito con una scheda a colori. Su un monitor a colori, determina la visualizzazione in bianco e nero.
\EDITOR (o\ED)	Richiama il programma MS-DOS Editor, che è un editore di testo a tutto schermo in grado di creare, modificare, salvare e stampare dei file di testo in formato ASCII.
\G	Nei monitor CGA determina l'aggiornamento più veloce possibile dello schermo. Non è supportata pienamente da tutti i sistemi.
\H	Visualizza sullo schermo il numero massimo di righe possibili.
\MBF	Fa sì che le funzioni di conversione incorporate in QBASIC (MKS$, MKD$, CVS e CVD) trattino i numeri in formato IEEE come se fossero in formato binario Microsoft (Microsoft Binary Format). In altri termini, le trasforma rispettivamente in MKSMBF$, MKDMBF$, CVSBMF e CVDMBF.
\NOHI	Permette di utilizzare un monitor che non supporta la visualizzazione ad alta intensità.
nome file	Carica in memoria ed esegue il programma "nome file" all'avvio di QBASIC. Se il programma è stato creato con GW-BASIC o con BASICA, è necessario che sia stato salvato con l'opzione A.

Tabella 1 *- Le opzioni di QBASIC che si possono richiamare quando si avvia il programma in ambiente DOS*

Indipendentemente dalla modalità di avvio, QBASIC presenta per prima cosa la videata di benvenuto di Figura 1, che propone subito una scelta tra:

- premere il tasto <Invio>, per accedere a una serie di schermate di aiuto (dette "Guida rapida")
- premere il tasto <Esc>, per cancellare la finestra centrale della videata ed entrare nell'"ambiente" di QBASIC.

Per adesso premiamo il tasto <Esc>.

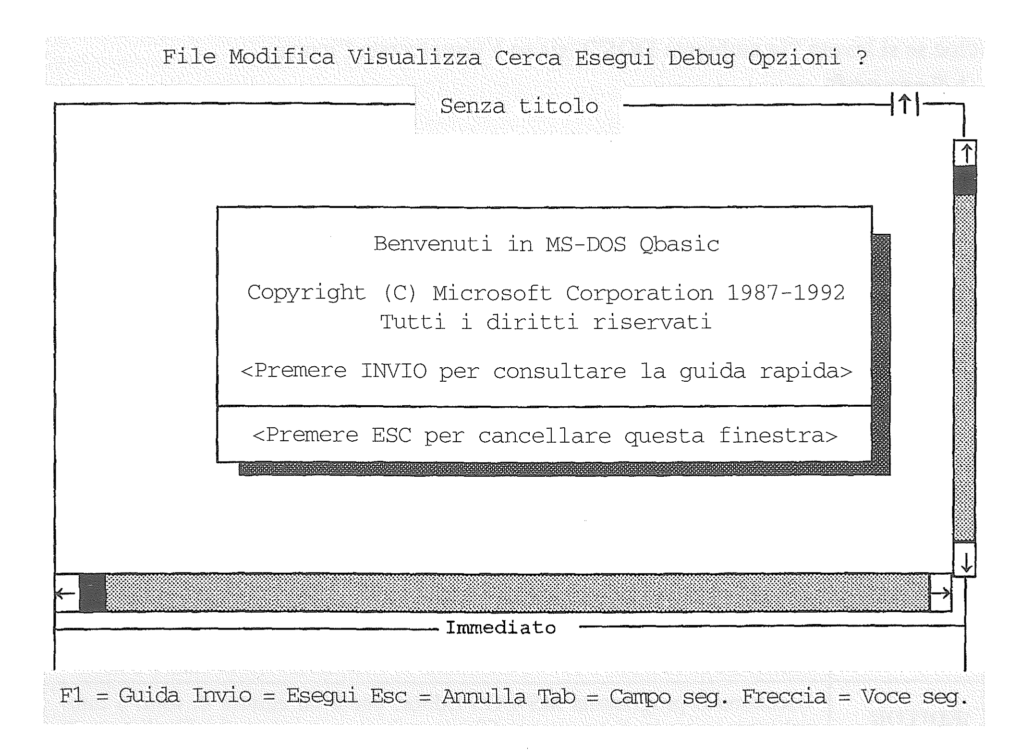


***Figura 1** - La videata di QBASIC con la finestra di benvenuto.*

LA VIDEATA DI QBASIC

Scompare la finestra di benvenuto e compare la videata di Figura 2, che contiene nella prima riga la *barra dei menu,* quindi due finestre intitolate rispettivamente

Senza titolo e Immediato

e nell'ultima riga la *barra di riferimento* e l'indicatore del numero di riga e di colonna del cursore. La finestra dove si trova il cursore lampeggiante è detta *finestra attiva,* e il suo titolo è visualizzato in modalità evidenziata. Il cursore può essere spostato da una finestra all'altra (facendola diventare attiva) in due modi:

- premendo il tasto funzione <F6>, oppure
- posizionando il puntatore del mouse nella finestra desiderata e premendo il suo pulsante attivo (operazione che d'ora in poi chiamerò "cliccare").

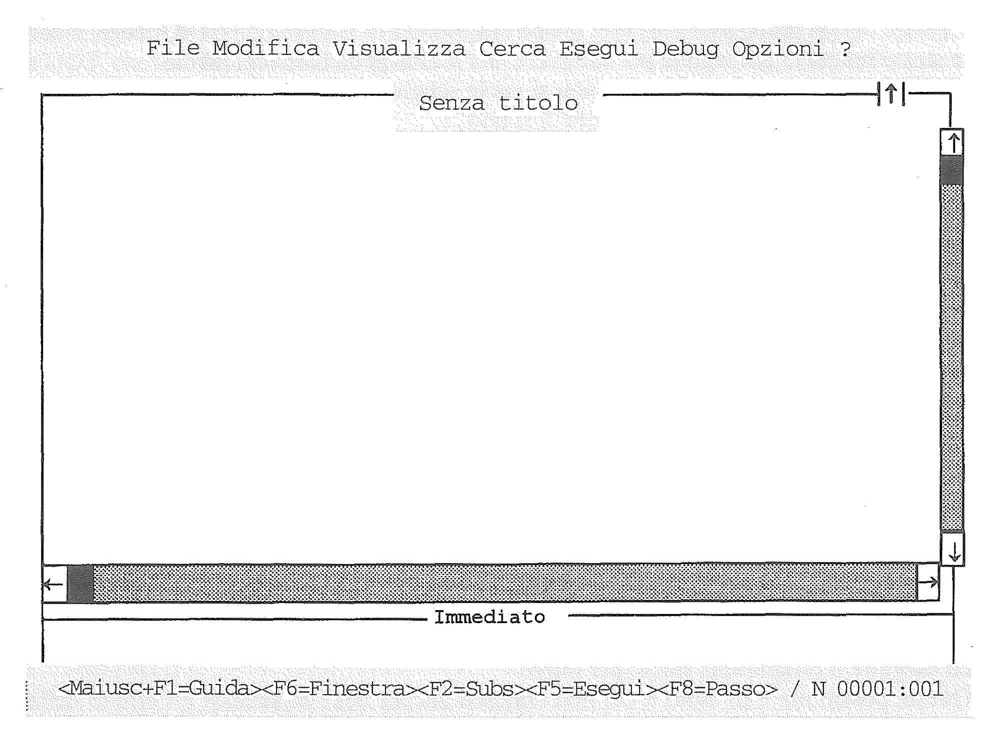


Figura 2 - La videata di QBASIC

LA BARRA DEI MENU

La **barra dei menu** presente nella prima riga di schermo

```
File  Modifica  Visualizza  Cerca  Esegui  Debug  Opzioni  ?
```

contiene i titoli di altrettanti menu visualizzabili, e può essere attivata/disattivata in due modi:

- premendo il tasto <Alt> (che si comporta come un interruttore), oppure
- posizionando il puntatore del mouse su uno dei suoi titoli e cliccando.

Se si preme il tasto <Alt> viene evidenziata la prima lettera di ogni titolo, e il primo titolo ("File") viene ulteriormente evidenziato con una barra scura. Il menu desiderato può essere allora aperto in due modi:

- digitando la lettera corrispondente, oppure
- portando la barra scura sul suo titolo, tramite i tasti di movimento del cursore, e premendo il tasto <Invio>.

Una volta aperto un menu, compaiono le sue voci, quelle attive con una lettera evidenziata, mentre la prima voce è evidenziata anche con una barra scura.

La voce desiderata si può scegliere anche ora usando la tastiera oppure il mouse, e con le stesse modalità già seguite per compiere una scelta nella barra dei menu.
Se invece si è aperto un menu, ma non si vuole scegliere in esso una particolare opzione (o, come si dice, lo si vuole "chiudere"), si può:

- premere il tasto <Esc>, oppure
- portare il puntatore in una zona dello schermo diversa dalle righe delle opzioni e cliccare.

L'ultimo titolo della barra dei menu è costituito da un

?

Esso permette di accedere alle stesse informazioni di aiuto visualizzabili, all'avvio di QBASIC con il tasto <Invio> o, durante la sua esecuzione, con la combinazione di tasti <Shift>+<F1>.

PERCHÉ DUE FINESTRE?

La ragione del fatto che QBASIC presenti due finestre ("Senza titolo" e "Immediato") è legata alla natura "interattiva" di questo linguaggio, cioè alla possibilità di provare l'effetto delle singole istruzioni che costituiscono un programma prima ancora di scrivere ed eseguire il programma stesso per intero.
Questa possibilità è di aiuto a chi non conosce ancora bene la sintassi e l'effetto delle varie istruzioni di QBASIC, e si realizza scrivendo le istruzioni da provare nella finestra "Immediato", anziché in quella "Senza titolo".
È possibile far estendere a tutto lo schermo la finestra "Senza titolo", con la conseguente scomparsa della finestra "Immediato", in due modi:

- premendo i tasti <Ctrl>+F10, oppure
- posizionando il puntatore del mouse sul simbolo |↑| (presente nella seconda riga) e cliccando.

Per ritornare alla situazione precedente, basta ripetere una di queste due operazioni.

LA FINESTRA "SENZA TITOLO"

Per vedere un esempio dei diversi modi di funzionamento delle due finestre, scriviamo nella finestra "Senza titolo" l'istruzione

```
print 5*8
```

e premiamo il tasto <Invio>.
Ci accorgiamo che non viene mostrato il risultato dell'operazione, ma viene posizionato il cursore nella seconda riga per l'immissione di un'altra istruzione. Se invece vogliamo vedere il risultato, dobbiamo eseguire questo programma (che è costituito da una sola istruzione), operando come segue:

- premere il tasto <F5>, oppure
- aprire il menu "Esegui" e scegliere l'opzione "Avvia".

La videata di QBASIC scompare momentaneamente, per ripresentare l'ultima videata del DOS, che termina ora con le righe

```
C:\>qbasic
 40

Premere un tasto per continuare
```

(Se QBASIC è stato avviato in ambiente Windows, non compare la riga `C:\>qbasic`).
Se si preme un tasto qualsiasi si riottiene l'ultima videata di QBASIC.

LA FINESTRA "IMMEDIATO"

Se invece scriviamo l'istruzione

```
print sqr(25)
```

nella finestra "Immediato" e premiamo il tasto <Invio>, otteniamo subito il risultato (la radice quadrata, o *sq*uare *r*oot di 25), visualizzato nella solita videata di MS-DOS, che adesso contiene le righe

```
C:\>qbasic
 40
  5
```

Anche adesso si riottiene l'ultima videata di QBASIC premendo un tasto qualsiasi.

L'ULTIMA RIGA

L'ultima riga della videata di QBASIC

```
<Maius+F1=Guida><F6=Finestra><F2=Subs><F5=Esegui><F8=Pas-so> N 00001:001
```

contiene la *barra di riferimento* e l'indicatore del numero di riga e di colonna in cui si trova il cursore.
La barra di riferimento indica alcuni tasti con la relativa operazione che compiono, e il suo aspetto cambia leggermente a seconda di quale sia la finestra attiva.
L'indicatore riserva 5 posizioni per la riga e 3 posizioni per la colonna in cui si trova il cursore, ed è utile soprattutto quando si scrive un programma nella finestra "Senza titolo".

Capitolo 3

Numeri e lettere



TIPI DI DATI NUMERICI

QBASIC permette di usare sia numeri senza il punto decimale, detti *interi,* sia numeri con il punto, detti *reali.* Entrambi i tipi si suddividono ulteriormente a seconda del campo di variabilità del numero, come indica la Tabella 2.

Tipo di dati	**Campo di variabilità**	
Intero	da -32 768	a 32 767
Intero LONG	da -2 147 483 648	a 2 147 483 647
Singola precisione	da $-3.402823*10^{38}$	a $3.402823*10^{38}$
Doppia precisione	da $-1.79769313486231*10^{308}$	a $1.79769313486231*10^{308}$

Tabella 2 *- Tipi di dati numerici di QBASIC*

NOTAZIONE SCIENTIFICA E FLOATING POINT

Per quanto riguarda la seconda colonna di Tabella 2, vi ricordo che i numeri molto grandi o molto piccoli sono scritti di solito nella **notazione scientifica,** nella quale ad esempio

$$3.402823*10^{38}$$

indica il numero 3.402823 con il punto decimale spostato a destra di 38 posizioni (naturalmente le posizioni vuote sono riempite di zeri).
Dalla notazione scientifica deriva la notazione **floating point** (che in italiano si traduce **virgola mobile,** in quanto gli inglesi separano i decimali con il punto, mentre noi con la virgola).
Nella notazione floating point il numero è rappresentato dalla sua *mantissa,* cioè le cifre che devono seguire il punto affinché la parte intera valga 0, seguita dall'*esponente* a cui si deve elevare il numero 10 per moltiplicare la mantissa.
Così il numero precedente si indica, in virgola mobile,

```
3402823 E+39
```

che significa

$$0.3402823*10^{39}$$

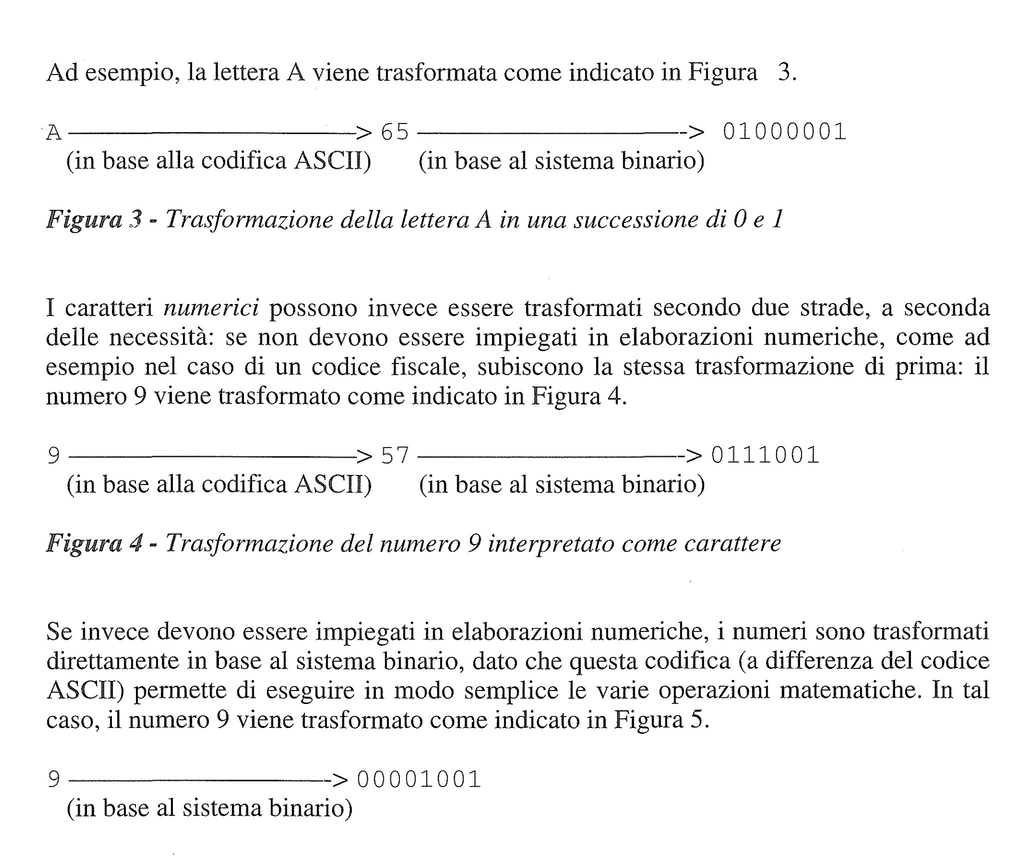
La notazione in virgola mobile permette di rappresentare numeri molto grandi o molto piccoli impiegando un numero ristretto di cifre (nell'esempio precedente, 7 per la mantissa una per il segno dell'esponente e 2 per il suo valore).
Naturalmente in tal modo si perde di precisione nel caso si debba operare con numeri costituiti da molte cifre significative, in quanto ad esempio, un numero quale 123456789 deve essere arrotondato a 123456800.

TABELLA DEI CODICI ASCII

Per scrivere un programma non è necessario avere una conoscenza approfondita della struttura fisica di un computer *(hardware);* in qualche caso è però necessario possedere semplici nozioni di hardware per capire la logica di alcuni aspetti della programmazione. È questo il caso della codifica ASCII. Vi ricordo che la memoria centrale di un computer si può considerare, dal punto di vista logico, come un insieme di tanti quadratini o *locazioni,* ciascuno dei quali è contrassegnato da un indirizzo e può contenere un carattere alfabetico, numerico o di altro tipo. Tuttavia, dal punto fisico, un computer è in grado di memorizzare, elaborare e trasmettere soltanto un'informazione di tipo binario, che convenzionalmente si indica con una successione di "0" e "1" *(bit).* Per questa ragione è necessario che il carattere contenuto in ogni locazione sia espresso come insieme di un certo numero di bit (si usano quasi sempre gruppi di 8 bit, detti *byte*), secondo una corrispondenza o codifica che in linea di principio potrebbe essere arbitraria, ma che di fatto segue dei criteri standard. Precisamente, i caratteri *alfanumerici* normalmente presenti su una tastiera sono dapprima codificati con un numero detto *codice ASCII,* come è indicato in Tabella 3 (in cui si vede che, ad esempio, il codice ASCII della lettera A è il numero 65); quindi tale numero viene trasformato in una successione di otto "0" e "1" secondo le regole del *sistema di numerazione binario.*

D	E		D	E		D	E		D	E		D	E		D	E		D	E		D	E	
00	00		32	20		64	40	@	96	60	`	128	80	Ç	160	A0	á	192	C0	└	224	E0	α
01	01	☺	33	21	!	65	41	A	97	61	a	129	81	ü	161	A1	í	193	C1	┴	225	E1	β
02	02	☻	34	22	"	66	42	B	98	62	b	130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
03	03	♥	35	23	#	67	43	C	99	63	c	131	83	â	163	A3	d	195	C3	├	227	E3	π
04	04	♦	36	24	$	68	44	D	100	64	d	132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
05	05	♣	37	25	%	69	45	E	101	65	e	133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
06	06	♠	38	26	&	70	46	F	102	66	f	134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
07	07	•	39	27	'	71	47	G	103	67	g	135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
08	08	◘	40	28	(	72	48	H	104	68	h	136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
09	09	○	41	29	)	73	49	I	105	69	i	137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
10	0A	◙	42	2A	*	74	4A	J	106	6A	j	138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
11	0B	♂	43	2B	+	75	4B	K	107	6B	k	139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
12	0C	♀	44	2C	,	76	4C	L	108	6C	l	140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
13	0D	♪	45	2D	-	77	4D	M	109	6D	m	141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
14	0E	♫	46	2E	.	78	4E	N	110	6E	n	142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
15	0F	☼	47	2F	/	79	4F	O	111	6F	o	143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
16	10	►	48	30	0	80	50	P	112	70	p	144	90	É	176	B0		208	D0	╨	240	F0	≡
17	11	◄	49	31	1	81	51	Q	113	71	q	145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
18	12	↕	50	32	2	82	52	R	114	72	r	146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
19	13	‼	51	33	3	83	53	S	115	73	s	147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
20	14	¶	52	34	4	84	54	T	116	74	t	148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
21	15	§	53	35	5	85	55	U	117	75	u	149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
22	16	▬	54	36	6	86	56	V	118	76	v	150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
23	17	↨	55	37	7	87	57	W	119	77	w	151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
24	18	↑	56	38	8	88	58	X	120	78	x	152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
25	19	↓	57	39	9	89	59	Y	121	79	y	153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
26	1A	→	58	3A	:	90	5A	Z	122	7A	z	154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
27	1B	←	59	3B	;	91	5B	[	123	7B	{	155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
28	1C	∟	60	3C	<	92	5C	\	124	7C	|	156	9C	£	188	BC	╝	220	DC	▄	252	FC	ⁿ
29	1D	↔	61	3D	=	93	5D	]	125	7D	}	157	9D	¥	189	BD	╜	221	DD	▌	253	FD	²
30	1E	▲	62	3E	>	94	5E	^	126	7E	~	158	9E	₧	190	BE	╛	222	DE	▐	254	FE	⌡
31	1F	▼	63	3F	?	95	5F	_	127	7F		159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	

***Tabella 3** - Tabella dei codici ASCII decimali (D) ed esadecimali (E).*
Dal codice 128 in poi è indicata la corrispondenza PC Internazionale



Ad esempio, la lettera A viene trasformata come indicato in Figura 3.

A ————————————> 65 ————————————> 01000001
(in base alla codifica ASCII) (in base al sistema binario)

Figura 3 - *Trasformazione della lettera A in una successione di 0 e 1*

I caratteri *numerici* possono invece essere trasformati secondo due strade, a seconda delle necessità: se non devono essere impiegati in elaborazioni numeriche, come ad esempio nel caso di un codice fiscale, subiscono la stessa trasformazione di prima: il numero 9 viene trasformato come indicato in Figura 4.

9 ————————————> 57 ————————————> 0111001
(in base alla codifica ASCII) (in base al sistema binario)

Figura 4 - *Trasformazione del numero 9 interpretato come carattere*

Se invece devono essere impiegati in elaborazioni numeriche, i numeri sono trasformati direttamente in base al sistema binario, dato che questa codifica (a differenza del codice ASCII) permette di eseguire in modo semplice le varie operazioni matematiche. In tal caso, il numero 9 viene trasformato come indicato in Figura 5.

9 ————————————> 00001001
(in base al sistema binario)

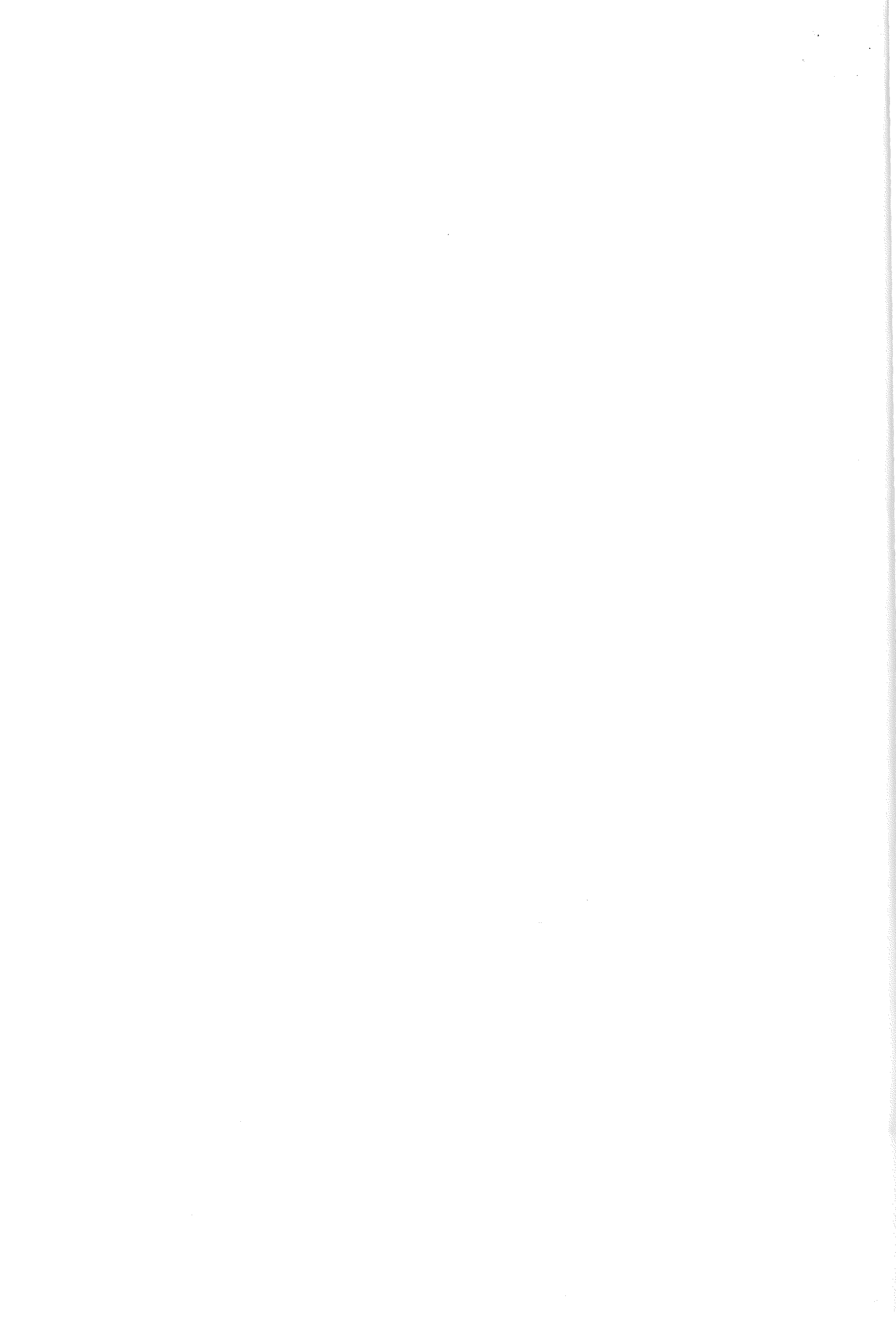
Figura 5 - *Trasformazione del numero 9 interpretato come numero*

Come si vede, è quindi possibile che una stessa sequenza di 0 e 1 indichi due caratteri differenti, e questa ambiguità viene risolta dai linguaggi di programmazione attraverso la *dichiarazione* del tipo delle variabili, come vedremo nel paragrafo "Dichiarazione del tipo di variabili" (→ pag. 21).



Capitolo 4

Variabili e costanti



CHE COSA SONO LE VARIABILI E LE COSTANTI

I moderni linguaggi di programmazione (e quindi anche QBASIC) permettono di fare riferimento alle varie quantità presenti nella memoria centrale di un computer semplicemente indicandole con un nome simbolico, anziché attraverso l'indirizzo della posizione in cui si trovano fisicamente.
In tal modo chi scrive un programma si può concentrare sulla sua logica, anziché sulla struttura del computer che lo dovrà eseguire.
Se la quantità indicata con il nome simbolico può cambiare durante l'esecuzione di un programma viene detta **variabile**, altrimenti **costante**; la differenza comunque è solo nell'utilizzo, dato che QBASIC tratta nello stesso modo variabili e costanti. In particolare, l'operazione che fa corrispondere un nome a una quantità si chiama **assegnazione** di valore o **inizializzazione** di una variabile.
Ad esempio, se intendiamo numerare le pagine di un documento, possiamo assegnare dapprima alla variabile PAGINA il valore 1 con un'istruzione del tipo

```
LET PAGINA = 1
```

e quindi aumentare tale valore con l'istruzione

```
PAGINA = PAGINA+1
```

Osserviamo che l'ultima uguaglianza rappresenterebbe un'equazione impossibile in matematica, mentre in programmazione significa semplicemente:

"assegna alla variabile PAGINA il valore attualmente posseduto dalla variabile PAGINA incrementato di 1".

Se invece vogliamo usare più volte un valore molto preciso di π (il rapporto fra la circonferenza e il diametro di un cerchio), ci conviene assegnare una volta per tutte tale valore a una costante, con un'istruzione del tipo

```
CONST PI = 3.141592653589#
```

e quindi usare nei nostri calcoli la costante PI anziché il suo valore (approssimato) 3.141592653589, evidentemente meno facile da ricordare e da scrivere.

ASSEGNAZIONE DI VALORI DA PROGRAMMA (LET, CONST)

Le due istruzioni

```
LET PAGINA = 1
```
e
```
CONST PI = 3.141592653589#
```

che abbiamo visto nel paragrafo precedente sono un primo esempio di istruzioni che assegnano un valore, rispettivamente a una variabile e a una costante da programma (altre possibilità sono costituite dalle istruzioni INPUT, LINE INPUT, INPUT$ e INKEY$ esaminate nel paragrafo "Immissione di dati dalla tastiera", → pag. 55 e dalle

READ...DATA esaminate nel paragrafo "Lettura e assegnazione di valori", → pag. 98).
Le istruzioni LET e CONST hanno la seguente forma sintattica:

```
[LET] variabile = espressione
[CONST] costante = espressione[, costante = espressione] ...
```

dove: le clausole LET e CONST sono facoltative (e di fatto vengono quasi sempre omesse); il nome simbolico della *variabile* e della *costante* può comprendere fino a 40 dei caratteri A-Z, 0-9, "." e deve iniziare con una lettera; *l'espressione* è una qualsiasi combinazione di costanti o variabili, dello stesso tipo del nome simbolico a cui è assegnata (questo punto sarà chiarito nel paragrafo successivo).
Osserviamo che l'istruzione di assegnazione non è un'uguaglianza in senso algebrico, per cui non è lecito scambiare l'ordine dei due termini e scrivere, ad esempio,

```
1 = PAGINA
```

anziché

```
PAGINA = 1.
```

TIPI DI VARIABILI

QBASIC permette di usare quattro tipi di variabili numeriche, corrispondenti ai tipi di dati numerici visti al paragrafo "Tipi di dati numerici" (→ pag. 13), e cioè di tipo *intero, intero LONG, singola precisione, doppia precisione,* oltre a variabili di stringa, cioè contenenti caratteri alfabetici o numerici, ma questi ultimi codificati in un formato non adatto per le elaborazioni numeriche.
Ora, dato che questi tipi di variabili hanno lunghezza in byte diversa (come indica la Tabella 4), e che uno o più byte possono rappresentare caratteri diversi, è necessario indicare al computer qual è l'interpretazione che deve essere data ai caratteri associati a un nome simbolico, cioè *dichiarare* quale tipo di dati è contenuto nelle variabili che intendiamo usare.

Tipo di variabile	Lunghezza
Intero	2
Intero LONG	4
Singola precisione	4
Doppia precisione	8
Stringa	variabile

Tabella 4 *- I tipi di variabili impiegati in QBASIC, con le rispettive lunghezze*

DICHIARAZIONE DEL TIPO DI VARIABILI (DEF, CLEAR)

In alcuni linguaggi di programmazione la dichiarazione del tipo di variabili e costanti è effettuata in modo *esplicito:* ad esempio, in Pascal un'istruzione del tipo

```
var rad1, rad2: real;
    cont, i: integer
```

ha l'effetto di dichiarare reali le variabili rad1 e rad2, e intere le variabili cont e i.
QBASIC permette di dichiarare il tipo di variabili sia in modo implicito sia esplicito. La dichiarazione *implicita* si effettua facendo seguire al nome della variabile un suffisso, come indicato in Tabella 5.

Tipo di variabile	Suffisso
Intero	%
Intero LONG	&
Singola precisione	! (o nessuno)
Doppia precisione	#
Stringa	$

Tabella 5 *- Suffissi che identificano i vari tipi di variabili QBASIC in modo implicito*

Ad esempio, l'istruzione di assegnazione vista in precedenza

```
PAGINA = 1
```

definisce contemporaneamente a singola precisione la variabile PAGINA e le assegna il valore 1. La dichiarazione *esplicita* si effettua tramite le istruzioni indicate in Tabella 6.

Tipo di variabile	Istruzione
Intero	DEFINT
Intero LONG	DEFLNG
Singola precisione	DEFSNG
Doppia precisione	DEFDBL
Stringa	DEFSTR

Tabella 6 *- Istruzioni che definiscono i vari tipi di variabili QBASIC in modo esplicito*

Queste istruzioni hanno forme sintattiche del tipo

```
DEFINT A-G
```

che dichiara intere le variabili i cui nomi inizino con una lettera compresa fra A e G, o del tipo

```
DEFDBL D-H, S-W
```

che dichiara in doppia precisione le variabili i cui nomi inizino con una lettera compresa fra D e H, oppure fra S e W.
La definizione implicita tramite suffisso ha sempre la precedenza sulla definizione esplicita tramite istruzione.
Osserviamo che la scelta del tipo di variabile dev'essere effettuata in modo da non sprecare inutilmente spazio in memoria; perciò sarà meglio scrivere l'istruzione

```
PAGINA% = 1
```

che impiega 2 byte per memorizzare il dato, e permette di numerare le pagine fino a 32767, piuttosto che

```
PAGINA = 1
```

che impiega 4 byte e permette di numerare le pagine fino a $3*10^{38}$. (valore decisionale fuori dal comune)
È anche possibile usare, nel corso di un programma, una variabile che non sia stata in precedenza inizializzata: in tal caso QBASIC pone uguale a zero il valore iniziale delle variabili numeriche e imposta le stringhe a lunghezza nulla.
Questa situazione è anche prodotta dall'istruzione

```
CLEAR
```

che fra gli altri effetti ha quello di cancellare dalla memoria del computer tutte le variabili.

Capitolo 5

Visualizzazione di testo sullo schermo



RIGHE, COLONNE E CELLE

QBASIC mantiene dalle precedenti versioni di BASIC la distinzione fra testo e grafica, nel senso che impiega istruzioni differenti per visualizzare caratteri alfanumerici e semigrafici (testo) oppure singoli puntini e figure geometriche (grafica); di quest'ultima ci interesseremo nel capitolo "La grafica" (→ pag. 147). Agli effetti della visualizzazione del testo, lo schermo si può considerare come una griglia di *righe* orizzontali e di *colonne* verticali; con la loro intersezione le righe e le colonne determinano delle *celle,* ciascuna delle quali visualizza al suo interno un carattere (vedi Figura 6). La configurazione standard dello schermo è di 80 colonne e 25 righe, ma sulle ultime due righe non è possibile scrivere. Infatti QBASIC impiega la 25ª riga per visualizzare il messaggio. `Premere un tasto per continuare` e la 24ª riga per separare questo messaggio dal resto del testo (l'argomento sarà ripreso dalla **Nota** del paragrafo "Stampa di grafici", → pag. 173).

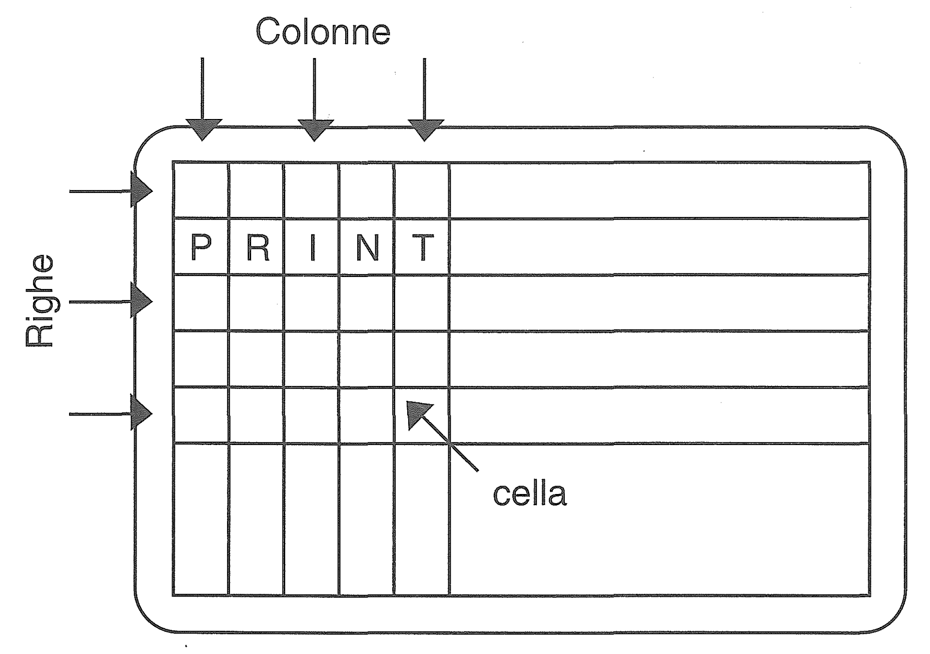


Figura 6 *- Colonne e righe nello schermo di testo*

Nota. Nel seguito del libro vedremo numerosi esempi di istruzioni o programmi adatti a illustrare le varie forme sintattiche e le tecniche di programmazione descritte; se al termine di una o più istruzioni compare il simbolo <F5> significa che, dopo aver copiato le istruzioni sulla videata di QBASIC, la pressione del tasto funzione <F5> permette di "provare" immediatamente l'effetto delle istruzioni, ottenendo sulla videata del DOS il risultato indicato dopo i caratteri <F5>. Ad esempio, la scrittura

```
PRINT "prova"                 <F5>
prova
```

significa che, se scriviamo sullo schermo di QBASIC l'istruzione

```
PRINT "prova"
```

e premiamo il tasto <F5>, sullo schermo del DOS compare la parola

```
prova
```

ISTRUZIONE PRINT

L'istruzione più comune per scrivere sullo schermo è

```
PRINT [lista di espressioni]
```

dove la *lista* è un elenco di una o più costanti o variabili numeriche o di stringa; nel caso di più *espressioni,* queste vanno separate con virgola (","), punto e virgola (";"), spazi vuoti o tabulazioni.
Se le *espressioni* sono separate con la virgola, vengono visualizzate ciascuna all'inizio di una delle sei zone di stampa (che iniziano rispettivamente alle colonne 0, 14, 28, 42, 56, 70) in cui è diviso lo schermo, come indicato in Figura 7. Esempio:

```
PRINT 5*3, 8-10  <F5>
 15           -2
```

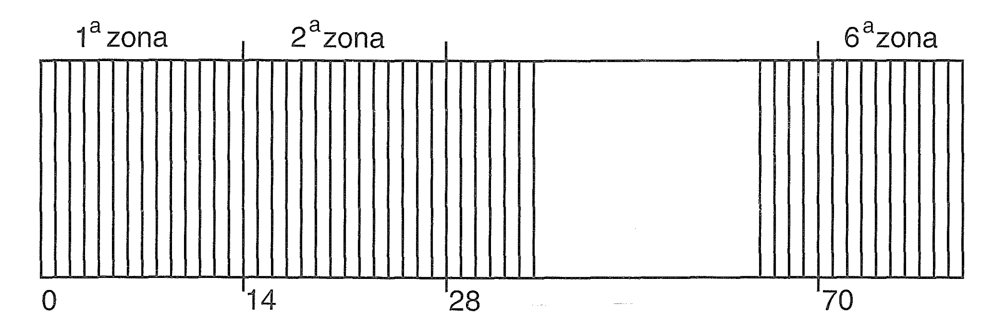


Figura 7 *- Suddivisione dello schermo in zone di stampa*

Se le *espressioni* della *lista* sono separate con punto e virgola (o con spazi vuoti o con tabulazioni, che vengono convertiti automaticamente in ";"), sono visualizzate una di seguito all'altra; tuttavia i valori numerici sono sempre preceduti da un carattere, che per i numeri positivi è il carattere vuoto, per quelli negativi il segno meno ("-"). Esempio:

```
PRINT 5*3; 8-10; "ferro"; "via"
PRINT "ferro", "via"          <F5>
 15 -2ferrovia
ferro         via
```

Se l'ultima espressione della *lista* è seguita da "," o da ";", questo carattere determinerà la posizione di stampa della prima espressione indicata nella successiva istruzione PRINT: in altri termini, la coppia di istruzioni

```
PRINT 5*3;
PRINT 8-10
```

ha lo stesso effetto dell'istruzione

```
PRINT 5*3; 8-10
```

Se si chiede a PRINT di visualizzare una linea più lunga di 80 caratteri (la larghezza dello schermo), la linea viene spezzata e visualizzata su due o più righe di schermo.
Se usata senza parametri, PRINT produce una riga vuota. Esempio:

```
PRINT "MENU"
PRINT
PRINT
PRINT "Capitolo 1°"
PRINT
PRINT "Capitolo 2°"                    <F5>

  MENU


  Capitolo 1°

  Capitolo 2°
```

La **cancellazione** di tutti i dati visualizzati sullo schermo si ottiene con l'istruzione

```
CLS
```

che in genere sarà bene far eseguire prima di qualsiasi istruzione che scriva sullo schermo.

VISUALIZZAZIONE DI DATI FORMATTATI (PRINT USING)

Per avere un maggiore controllo sul formato con cui sono visualizzati i dati, si usa l'istruzione PRINT USING anziché PRINT. Questa istruzione ha la seguente forma sintattica

PRINT USING *stringa di formato; lista di espressioni*

dove: la *stringa di formato* è costituita da 1 fino a 24 dei codici di controllo indicati nei due paragrafi successivi racchiusi tra virgolette, e le *espressioni* possono essere, come nell'istruzione PRINT, numeriche o di stringa.
Nel caso di espressioni di stringa, le possibilità di controllo permettono di stampare le espressioni limitatamente al primo (!), o ai primi (\...\) caratteri o per intero (&). Maggiori possibilità di controllo si hanno invece per le espressioni numeriche.

CODICI DI CONTROLLO PER ESPRESSIONI DI STRINGA

!

Visualizza solo il primo carattere delle *espressioni* successive. Esempio:

```
A$ = "CAPO": B$ = "SEZIONE"
PRINT USING "!"; A$; B$                    <F5>
CS
```

\\

Visualizzano solo i primi due caratteri delle *espressioni* successive; se tra le barre è inserito un carattere vuoto (\ \), visualizzano i primi tre caratteri, se sono inseriti due caratteri vuoti (\ \) i primi quattro caratteri, e così via. Se un'espressione è più lunga della *stringa,* i caratteri in eccedenza sono ignorati; se invece è più corta, viene allineata a sinistra e alla sua destra sono inseriti spazi vuoti. Esempi:

```
A$ = CAPO": B$ = "SEZIONE"
PRINT USING "\\"; A$; B$         <F5>
CASE

PRINT USING "\   \"; A$; B$      <F5>
CAPO SEZIO
```

&

Visualizza la *stringa* esattamente come è stata inserita (cioè come farebbe un'istruzione PRINT).

_

Visualizza il codice di controllo che lo segue come un carattere alfabetico. Esempio:

```
PRINT USING "_&&"; Company      <F5>
&Company
```

In effetti, l'unico modo per stampare davanti a un'*espressione* uno dei caratteri "_" "!" "\" "&" è di iniziare la *stringa di formato* rispettivamente con "__" "_!" "_\" "_&".

CODICI DI CONTROLLO PER ESPRESSIONI NUMERICHE

#
Visualizza una cifra del numero per ogni carattere "#" presente nella *stringa*. Se il numero ha meno cifre dei caratteri "#" specificati, viene allineato a destra e alla sua sinistra sono inseriti spazi vuoti; se invece ha più cifre, viene arrotondato. Esempio:

```
PRINT USING "###.##"; 123.4567
PRINT USING "###.##"; 1.2345          <F5>
123.46
  1.23
```

**
Visualizzano uno o più asterischi davanti ai numeri, in sostituzione degli eventuali spazi vuoti. Esempio:

```
PRINT USING "**#.##"; 123.4567
PRINT USING "**#.##"; 1.2345          <F5>
123.46
**1.23
```

(osserviamo che anche un carattere "*", come pure un "#", contribuisce a determinare il numero di cifre da visualizzare).

+
Visualizza il segno ("+" o "-") davanti ai numeri, se è posto all'inizio della *stringa di formato;* se è posto alla fine, visualizza il segno dopo i numeri. Esempi:

```
PRINT USING "+###.##"; 123.4567; -123.4567  <F5>
+123.46-123.46
```

```
PRINT USING "###.##+"; 123.4567; -123.4567  <F5>
123.46+123.46-
```

- (meno)
Visualizza il segno "-" dopo i numeri negativi, se è posto alla fine della *stringa di formato.* Esempio:

```
PRINT USING "###.##-"; 123.4567; -123.4567  <F5>
123.46 123.46-
```

. (punto)
Visualizza il punto decimale nella posizione in cui si trova.

$ (oppure $$)
Visualizza il simbolo "$" davanti ai numeri. Esempio:

```
PRINT USING "$###.##"; 123.4567                   <F5>
$123.46
```

^^^^
Visualizzano i numeri nella forma in virgola mobile (→ pag. 13) E+xx. Esempio:

```
PRINT USING "#.####^^^^"; 123.4567             <F5>
0.1235E+03
```

^^^^^
Visualizzano i numeri nella forma in virgola mobile E+xxx. Esempio:

```
PRINT USING "#.####^^^^^"; 123.4567           <F5>
0.1235E+003
```

Se all'inizio della *stringa di formato* sono presenti caratteri diversi dai codici di controllo, questi sono stampati come scritti. Esempio:

```
PRINT USING "Formato con 3 decimali ###.###"; 123.4567 <F5>
Formato con 3 decimali 123.457
```

SALTO DI SPAZI (SPC, SPACE$)

Se in un'istruzione PRINT si inserisce la funzione

```
SPC(n)
```

vengono saltati *n* spazi in una riga di stampa, come mostra il seguente esempio (nel quale la stringa 1234567890 è stata inserita solo per fornire un controllo sul numero di spazi stampati):

```
PRINT "1234567890"
PRINT "Nome"; SPC(3); "Cognome"                  <F5>
 1234567890
 Nome   Cognome
```

Lo stesso effetto si ottiene anche usando la funzione

```
SPACE$(n)
```

che genera una stringa di *n* spazi, e può essere usata anche al di fuori di un'istruzione PRINT. Esempio:

```
S3$= SPACE$(3)
PRINT "Nome"; S3$; "Cognome"                      <F5>
 Nome   Cognome
```

AVANZAMENTO A UNA COLONNA SPECIFICA (TAB)

L'uso della funzione

```
TAB(n)
```

in un'istruzione PRINT permette di saltare alla *n*-ma colonna di schermo in una riga di stampa. Esempio:

```
PRINT "1234567890"
PRINT "Nome"; TAB(10); Cognome    <F5>
1234567890
Nome        Cognome
```

La differenza tra le funzioni SPC e SPACE$ da una parte, e TAB dall'altra, è che le prime determinano uno spostamento *relativo* all'ultima posizione di stampa, la seconda uno spostamento *assoluto* rispetto al margine sinistro dello schermo.

POSIZIONAMENTO DEL CURSORE (LOCATE, CSRLIN, POS)

Un controllo ancora maggiore della posizione del cursore di quello permesso dalle funzioni SPC, SPACE$ e TAB è fornito dall'istruzione

```
LOCATE [riga][,[colonna][,[cursore][,[inizio][,fine]]]]
```

che permette di posizionare il cursore nella *riga* e *colonna* di schermo indicate. Il valore di *riga* deve essere compreso fra 1 e 25, quello di *colonna* fra 1 e 40 oppure fra 1 e 80, a seconda della larghezza dello schermo (vedi la successiva istruzione WIDTH). Ad esempio, la seguente coppia di istruzioni

```
LOCATE 12, 30
PRINT "Centro dello schermo"
```

visualizza la frase "Centro dello schermo" al centro dello schermo. Il cursore risulta visibile se il parametro *cursore* vale 1, invisibile se vale 0. Se è visibile, il cursore ha la forma di un rettangolo, la cui altezza è determinata dai valori delle righe di *inizio* e di *fine* nella matrice in cui sono ricavati i caratteri (vedi il successivo capitolo "La grafica," a pag. 147).

0 inizio

7 fine

Per conoscere la posizione del cursore sullo schermo si usano le funzioni

```
CSRLIN
```

che fornisce la riga, e

```
POS(1)
```

che fornisce la colonna attuale del cursore (l'argomento 1 è completamente irrilevante, e può essere sostituito da altro numero intero). Esempio:

```
LOCATE 5, 20
PRINT CSRLIN; LOC(1)          <F5>
                5 23
```

CAMBIAMENTO DEL NUMERO DI COLONNE O DI RIGHE (WIDTH)

È possibile cambiare il numero delle colonne visualizzate sullo schermo e, se il computer ha una scheda EGA o VGA (vedi il successivo paragrafo "Schede grafiche", a pag. 147), anche delle righe usando l'istruzione

```
WIDTH [colonne][, righe]
```

dove le *colonne* possono essere 40 o 80, e le *righe* 25, 30, 43, 50 o 60.

STAMPA SU CARTA (LPRINT, LPRINT USING)

Le istruzioni PRINT e PRINT USING viste nei paragrafi precedenti sono simili alle istruzioni LPRINT e LPRINT USING, che permettono di trasferire i risultati delle elaborazioni alla stampante anziché allo schermo. Esse hanno la stessa forma sintattica, e cioè

```
LPRINT [lista di espressioni]
LPRINT USING stringa di formato; lista di espressioni
```

con gli stessi significati ed effetti dei parametri di PRINT e PRINT USING.
Anche in questo caso si assume che la stampante permetta di scrivere 80 caratteri per riga, ma tale numero può essere reimpostato, se la stampante permette di scrivere più di 80 caratteri per riga, con l'istruzione WIDTH.

FINESTRA DI TESTO (VIEW PRINT, CLS 2)

Finora la visualizzazione del testo è avvenuta sull'intero schermo. Tuttavia l'istruzione

VIEW PRINT [*lineasup* TO *lineainf*]

permette di ridurre l'area di visualizzazione a una finestra di testo, consistente in una porzione orizzontale di schermo compresa tra la *lineasup* e la *lineainf.* Una finestra di testo permette anche di controllare lo scorrimento verticale del testo, che scompare dalla parte superiore dello schermo quando è stata completata l'ultima riga di schermo. Invece, dopo un'istruzione VIEW PRINT, lo scorrimento ha luogo solo fra la *lineasup* e la *lineainf* della finestra, mentre l'istruzione

CLS 2

pulisce solo il testo all'interno della finestra, lasciando inalterato quello al suo esterno. Gli effetti dell'istruzione VIEW PRINT si possono esaminare eseguendo il seguente programma, che produce l'uscita di Figura 8.

```
CLS
LOCATE 3, 1
PRINT "Testo sopra la finestra. Non scorre"
LOCATE 4, 1:  PRINT "____________________"
LOCATE 11, 1: PRINT "____________________"
PRINT "Testo sotto la finestra."
PRINT "Premere un tasto per cancellarla"
VIEW PRINT 5 TO 10
FOR i = 1 TO 20
 PRINT i; "linea di testo"
 FOR k = 1 TO 500: NEXT
NEXT
DO: LOOP WHILE INKEY$ = ""
CLS 2:    END
```

Anche se alcune delle istruzioni di questo programma saranno spiegate più avanti, esso risulta abbastanza chiaro, e potrà essere modificato facilmente per le varie esigenze.

```
Testo sopra la finestra
_______________________
16 linea di testo
17 linea di testo
18 linea di testo
19 linea di testo
20 linea di testo
_______________________
Testo sotto la finestra
Premere un tasto per cancellarla
```

***Figura 8** - Creazione di una finestra di testo, con scorrimento e cancellazione indipendenti dal resto dello schermo*

Capitolo 6

Funzioni ed espressioni numeriche



CHE COS'È UNA FUNZIONE

Una funzione di QBASIC è una serie di istruzioni che forniscono un risultato utilizzabile da un programma o visualizzabile sullo schermo. Una funzione opera in genere su un'espressione numerica o di stringa, che viene detta il suo *argomento* o *parametro* ed è scritta tra parentesi tonde dopo il nome della funzione.
QBASIC possiede un certo numero di funzioni *predefinite* o *incorporate,* cioè progettate dagli autori del linguaggio e codificate una volta per tutte nell'interprete di QBASIC; esse si dividono in funzioni *numeriche,* e *di stringa.* QBASIC permette inoltre all'utente di definire proprie funzioni, come vedremo nel paragrafo "Funzioni" (→ pag. 97).
Riporto di seguito l'elenco delle funzioni numeriche e di stringa con semplici esempi del loro funzionamento, mentre illustrerò le funzioni necessarie per la gestione dei file nel capitolo che tratta di questo argomento.

FUNZIONI NUMERICHE

Le funzioni numeriche sono quelle che forniscono come risultato un numero. Il loro argomento può essere una variabile o costante numerica - che negli esempi che seguono è indicata con una delle lettere *x, y, z,* se si tratta di un valore reale, con *n* se intero - oppure una stringa di caratteri; nel secondo caso la funzione trasforma la stringa in numero.

ABS(*x*)
Fornisce il valore assoluto di *x,* cioè *x* stesso se è positivo, il suo opposto se è negativo. Esempio:

```
PRINT ABS(34.5) ; ABS(-5.8)        <F5>
  34.5      5.8
```

ATN(*x*)
Fornisce l'angolo, misurato in radianti, la cui tangente è *x.* Esempio:

```
PRINT ATN(1.55741)                 <F5>
  1.000001
```

Ricordo che:

- l'angolo di 1 radiante è quello formato da due raggi di una circonferenza che intercettano sulla circonferenza un arco lungo quanto il raggio (vedi Figura 9)

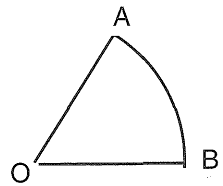


Figura 9 *- Se i segmenti AO, OB e l'arco AB sono uguali, l'angolo AOB è di 1 radiante*

* per convertire i radianti in gradi si moltiplica il numero di radianti per 180/3.141592653589.
* la tangente dell'angolo AOH in Figura 10 è data dal rapporto fra l'altezza AH e la base OH

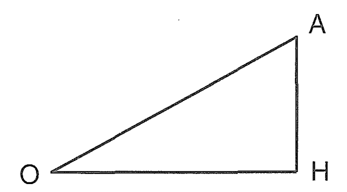


Figura 10 *- La tangente dell'angolo AOH è definita dal rapporto AH/OH*

CDBL(x)
Converte l'espressione numerica x in un valore in doppia precisione. Esempio:

```
PRINT 1/3, CDBL(1/3)                <F5>
 .3333333          .3333333333333333
```

CINT(x)
Converte x in un intero, arrotondando la parte decimale. L'argomento dev'essere compreso fra -32768 e 32767. Esempio:

```
PRINT CINT(56.78), CINT(-2.89)     <F5>
  57  -3
```

CLNG(x)
Converte x in un intero LONG (→ pag. 13) di 4 byte, arrotondando la parte decimale. L'argomento dev'essere compreso fra -2147483648 e 2147483647. Esempio:

```
PRINT CLNG(338457.8)                <F5>
  338458
```

COS(x)
Fornisce il coseno dell'angolo di ampiezza x radianti (ricordo che per convertire i gradi in radianti si moltiplica il numero di gradi per 3.141592653589/180). Esempio:

```
PRINT COS(1.047197)                 <F5>
  0.5000005
```

CSNG(x)
Converte x in un numero a precisione singola, arrotondando la parte decimale. Esempio:

```
PRINT CSNG(975.3421515#)            <F5>
  975.3422
```

EXP(x)
Fornisce il valore di e^x, dove e rappresenta il numero di Nepero, uguale a 2.7182818284590... Esempio:

```
PRINT EXP(2)                      <F5>
  7.389056
```

FIX(*x*)
Tronca *x* a un intero (senza arrotondare la parte decimale). Esempio:

```
PRINT FIX(56.78), FIX(-2.89)      <F5>
  56  -2
```

INT(x)
Fornisce l'intero uguale o immediatamente inferiore a *x*. Esempio:

```
PRINT INT(56.78), INT(-2.89)      <F5>
  56  -3
```

LOG(*x*)
Fornisce il logaritmo di *x* in base e, dove e rappresenta il numero di Nepero, uguale a 2.7182818284590... Esempio:

```
PRINT LOG(7.389056)               <F5>
  2
```

RND(*x*)
Fornisce un numero casuale in precisione semplice compreso tra 0 e 1, generato in base al valore del parametro *x*, secondo quanto indicato in Tabella 7.

Valore di x	Numero casuale fornito da RND
minore di 0	un numero diverso per ogni valore diverso di *x*
maggiore di 0 (o nessun valore)	il numero successivo di una sequenza predefinita
0	l'ultimo numero fornito

Tabella 7 *- Effetti del parametro* x *nella funzione RND*(x)

Consideriamo i seguenti esempi:

```
PRINT RND(-1), RND(-2), RND(-1)  <F5>
  .224007  .7133257   .2240007

PRINT RND(1), RND(0), RND(1)      <F5>
  .7055475       .7055475              .533424
```

I numeri casuali generati con valori di *x* maggiori di 0 appartengono a una sequenza predefinita, quindi sono in realtà pseudo-casuali. Si può ottenere una diversa sequenza di numeri pseudo-casuali facendo precedere la funzione RND dall'istruzione

```
RANDOMIZE n
```

che a ciascun valore dell'argomento *n* fa corrispondere una diversa sequenza di numeri. Un argomento a sua volta casuale è costituito dalla funzione TIMER (→ pag. 54), che fornisce il numero di secondo trascorsi dalla mezzanotte. Si può provare con i seguenti comandi:

```
PRINT RND (1), RND (1)
RANDOMIZE 5
PRINT RND (1), RND (1)              <F5>
 .7055475   .533424
 2.260333E-02           .1790583
```

SIN(*x*)
Fornisce il seno dell'angolo di ampiezza *x* radianti (ricordo che per convertire i gradi in radianti si moltiplica il numero di gradi per 3.141592653589/180). Esempio:

```
PRINT SIN(.5235988)                <F5>
  0.5
```

SGN(*x*)
Indica il segno di *x*, fornendo -1 se *x* è negativo, 0 se *x* è uguale a zero, 1 se *x* è positivo. Esempio:

```
PRINT SGN(-5), SGN(0), SGN(10)    <F5>
-1          0           1
```

SQR(*x*)
Fornisce la radice quadrata dell'argomento, purché *x* sia positivo o nullo. Esempio:

```
PRINT SQR(144)                     <F5>
  12
```

QBASIC non dispone di funzioni che forniscano radici di indice superiore a 2 (radici cubiche, quarte, ...); tuttavia esse si possono ottenere elevando il radicando a un opportuno esponente frazionario (rispettivamente 1/3, 1/4, ...). Ad esempio, la radice cubica di 8 si può ottenere con il comando

```
PRINT 8^(1/3)                      <F5>
   2
```

TAN (*x*)
Fornisce la tangente trigonometrica dell'angolo di ampiezza *x* radianti (ricordo che per convertire i gradi in radianti si moltiplica il numero di gradi per 3.141592653589/180). Esempio:

```
PRINT TAN(45*3.141593/180)  <F5>
  1
```

ESPRESSIONI NUMERICHE

Le variabili, le costanti e le funzioni numeriche possono essere collegate fra loro dagli *operatori aritmetici* indicati in Tabella 8 per formare delle espressioni numeriche, secondo le usuali regole algebriche.

Operatore	**Operazione**
^	Elevamento a potenza
*	Moltiplicazione
/	Divisione
\	Divisione intera
MOD	Modulo
*	Addizione
-	Sottrazione

***Tabella 8** - Operatori aritmetici di QBASIC*

Tra le operazioni indicate, quelle di divisione intera e di modulo possono essere poco familiari.
La **divisione intera** fornisce il risultato intero della divisione di due interi. Esempio:

```
PRINT 10 \ 3          <F5>
 3
```

Il **modulo** fornisce il resto della divisione di due interi. Esempio:

```
PRINT 10 MOD 3        <F5>
 1
```

È da tenere presente che, mentre nella scrittura algebrica un'espressione numerica può avere un aspetto *bidimensionale* (cioè occupare più di una riga), in QBASIC essa va scritta in modo *unidimensionale* (cioè su una sola riga). Ciò si ottiene con un uso opportuno delle parentesi (sempre tonde), come mostrano gli esempi di Tabella 9.

Scrittura algebrica	**Scrittura QBASIC**
$\frac{(1+i)^n - 1}{i}$	((1+I)^N-1)/I
$\frac{-b - \sqrt{b^2 - 4ac}}{2a}$	(-B-SQR(B^2-4*A*C))/(2*A)

***Tabella 9** - Differenti modi di scrivere un'espressione in algebra e in QBASIC*



Capitolo 7

Stringhe di caratteri

CHE COS'È UNA STRINGA

Una stringa è una successione continua di caratteri, quali le lettere dell'alfabeto, i segni di interpunzione e quelli matematici. QBASIC permette di trattare sia costanti sia variabili di stringa.

DICHIARAZIONE DI COSTANTI E VARIABILI (CONST, DEFSTR, AS STRING)

Una **costante** di stringa si può dichiarare in uno dei seguenti modi:

- Racchiudendo una sequenza di caratteri fra virgolette, come nella seguente istruzione PRINT:

  ```
  PRINT "Elaborazione del file. Attendere prego"
  ```

  Questa è detta "costante di stringa letterale".

- Ponendo un nome uguale a una stringa letterale, come nella seguente istruzione CONST:

  ```
  CONST Messaggio = "Controllare la stampante"
  ```

  Questa è detta "costante di stringa simbolica".

  Una **variabile** di stringa si può dichiarare in uno dei seguenti modi:

- Aggiungendo il suffisso di stringa ($) a un nome di variabile, come ad esempio in:

  ```
  LINE INPUT "Scrivi il tuo nome: "; Nome$
  ```

- Usando l'istruzione

  DEFSTR *lettera*

  che fa considerare stringhe tutte le variabili il cui nome inizi con *lettera,* a meno che esse terminino con uno dei suffissi dei tipi numerici (%, &, !, #). Se ad esempio si scrivono le due istruzioni seguenti:

  ```
  DEFSTR M
  Messaggio = "Sportello del drive aperto"
  ```

  la variabile Messaggio viene considerata di stringa.

- Dichiarando il nome della stringa in un'istruzione

  ```
  AS STRING
  ```

  come nella seguente:

  ```
  DIM Messaggio AS STRING
  ```

STRINGHE DI LUNGHEZZA VARIABILE E FISSA (DIM, RSET)

Le precedenti versioni del BASIC permettevano di utilizzare solo stringhe di **lunghezza variabile**, che cioè assumevano la lunghezza del numero di caratteri contenuti (da 0 a 32767). Ad esempio, una variabile così definita

```
A$ = "1234"
```

risultava lunga 4 caratteri, mentre una così definita

```
B$ = "123456"
```

risultava di lunghezza 6.
Oltre a mantenere le stringhe di lunghezza variabile, QBASIC dispone anche delle stringhe di **lunghezza fissa,** che vengono dichiarate con un'istruzione del tipo

```
DIM Messaggio AS STRING * 10
```

(che definisce la variabile Messaggio come una stringa di lunghezza 10).
Se a una stringa di lunghezza fissa si assegna un valore maggiore della sua lunghezza, il valore viene troncato alla destra. Si considerino ad esempio le seguenti istruzioni

```
DIM Messaggio AS STRING * 10
Messaggio = "Sportello del drive aperto"
PRINT Messaggio              <F5>
  Drive aper
```

Se a una stringa di lunghezza fissa si assegna un valore inferiore alla sua lunghezza, i caratteri mancanti sono riempiti con spazi vuoti, come mostrano le seguenti istruzioni

```
DIM Messaggio AS STRING * 10
Messaggio = "Fine"
PRINT Messaggio; Messaggio <F5>
  Fine      Fine
```

Come mostra l'ultimo esempio, i valori assegnati a variabili di lunghezza fissa sono automaticamente giustificati sulla sinistra. Se fosse necessario giustificarli sulla destra, si impiegherebbe l'istruzione

```
RSET
```

come mostra il seguente esempio (dove la linea di numeri è stata scritta solo per fornire un riferimento visivo)

```
DIM Messaggio AS STRING * 10
RSET Messaggio = "Fine"
PRINT "1234567890"
PRINT Messaggio              <F5>
  1234567890
        Fine
```

CONCATENAZIONE DI STRINGHE E DI CARATTERI (STRING$)

Le costanti e variabili di stringa possono essere combinate con l'operatore di concatenazione "+", confrontate con gli operatori di relazione e subire vari tipi di elaborazione con le funzioni di stringa.
Se due stringhe sono combinate con l'operatore "+" la seconda è aggiunta alla prima, come mostra il seguente esempio:

```
A$ = "capo"
B$ = "stazione"
C$ = A$ + B$
PRINT C$                    <F5>
 capostazione
```

Dato che la concatenazione non introduce spazi vuoti, se questi servono devono essere aggiunti nelle stringhe su cui si opera, ad esempio con definizioni del tipo

```
A$ = "capo "
B$ = "operaio"
```

oppure

```
A$ = "capo"
B$ = " operaio"
```

In tal modo, la coppia di istruzioni

```
C$ = A$ + B$
PRINT C$
```

produce l'uscita

```
  capo operaio
```

Lo stesso effetto si ottiene con le istruzioni

```
DIM A AS STRING * 5
DIM B AS STRING * 7
A = capo
B = operaio
PRINT A + B
```

Per concatenare dei caratteri tutti uguali tra loro, si può usare la funzione

STRING$(*m*, *n*)

che genera una stringa lunga *m* caratteri, tutti di codice ASCII (→ pag. 14) *n*. I caratteri presenti sulla tastiera possono essere indicati tra virgolette, anziché attraverso il numero *n*, come nel seguente esempio:

```
ASTE$ = STRING$(15, "*")
```

In questo caso la stringa di 15 asterischi generata da STRING$ è stata assegnata alla variabile ASTE$; un'altra possibilità è quella di stampare la stringa con un'istruzione PRINT, come nell'esempio seguente:

```
PRINT STRING$(10,64)  <F5>
  @@@@@@@@@@
```

Per concatenare degli spazi, anziché la funzione STRING$, si può usare la funzione

```
SPACE$(n)
```

che genera una stringa di *n* spazi.

CONFRONTO DI STRINGHE

Due stringhe possono essere confrontate raccordandole con uno degli *operatori di relazione* indicati in Tabella 10

Operatore	Significato
=	uguale
<>	diverso
<	minore
<=	minore o uguale
>	maggiore
>=	maggiore o uguale

***Tabella 10** - Operatori di relazione usati in QBASIC*

Nel confronto fra due caratteri, QBASIC considera "maggiore" quello che ha il codice ASCII maggiore.
Nel confronto fra due stringhe, QBASIC considera i valori ASCII dei caratteri che si corrispondono; il primo carattere in cui le due stringhe differiscono determina il loro ordine alfabetico. Ad esempio, le due stringhe "autore" e "autolavaggio" hanno uguali i primi quattro caratteri; poiché il valore ASCII di "r" è maggiore di quello di "l", risulta

```
autore > autolavaggio
```

Se tutti i caratteri di due stringhe si corrispondono, ma una è più lunga, essa viene considerata maggiore, per cui risulta, ad esempio:

```
"aaaaaa" > "aaa"
```

Osserviamo che nella Tabella dei codici ASCII di pag. 14 le lettere maiuscole precedono le minuscole, per cui risulta, ad esempio,

```
"ASCII" < "ascii"
```

Il confronto di due stringhe è impiegato di solito per costruire espressioni booleane (che vedremo a pag. 67) caratterizzate da un valore "vero" o "falso". Così, ad esempio, le espressioni

```
"abc" = "abc"
"aaaaaa" > "aaa"
```

sono entrambe vere.

Nota. Nel confronto fra due stringhe, gli spazi iniziali o finali risultano significativi. Perciò, ad esempio, la stringa

```
"    prova"
```

risulta minore della stringa

```
"prova"
```

dato che lo spazio ha un codice minore di "p". Invece la stringa

```
"prova     "
```

risulta maggiore della stringa

```
"prova"
```

Di questo fatto è bene tener conto quando si confrontano stringhe di lunghezza fissa (che possono contenere spazi alla fine) e stringhe di lunghezza variabile, come vedremo al successivo paragrafo, a proposito della funzione RTRIM$.

FUNZIONI DI STRINGA

Le funzioni di stringa sono quelle che forniscono come risultato una stringa di caratteri. Hanno come argomento una variabile o una costante di stringa, che negli esempi che seguono è indicata con *a$*, oppure un numero che trasformano in stringa.

ASC(*a$*), CHR$(*n*)
Sono due funzioni l'una inversa dell'altra: ASC fornisce il codice ASCII del primo carattere della stringa *a$*. Esempio:

```
PRINT ASC ("Q")                    <F5>
  81
```

CHR$ converte il numero intero *n* nel carattere che gli corrisponde secondo il codice ASCII. Esempio:

```
PRINT CHR$(81)                      <F5>
  Q
```

HEX$(*x*), OCT$(*x*)
HEX$(*x*) fornisce una stringa che rappresenta la conversione nel sistema esadecimale dell'espressione numerica *x*.
OCT$(*x*) fornisce una stringa che rappresenta la conversione nel sistema ottale di *x*. Se *x* ha un valore decimale, viene arrotondata a un intero. Esempio:

```
PRINT HEX$(32), OCT$(16)           <F5>
 20      20
```

INSTR
Un'operazione molto frequente che si compie sulle stringhe è la ricerca di una (sotto) stringa all'interno di un'altra stringa. La funzione

```
INSTR(a1$, a2$)
```

dice se la stringa *a2$* è contenuta nella stringa *a1$*, fornendo la posizione del primo carattere di *a1$* (se c'è) in cui comincia l'identità. Se invece *a2$* non è contenuta in *a1$*, INSTR fornisce il valore 0. Esempio:

```
A1$ = "capostazione"
A2$ = "posta"
PRINT INSTR(A1$, A2$)               <F5>
  3
```

In questa variante dalla precedente forma sintattica

```
INSTR(inizio, a1$, a2$)
```

la funzione INSTR permette invece di indicare da dove si vuole che inizi la ricerca in *a1$*. Così si avrebbe, ad esempio:

```
A1$ = "capostazione"
A2$ = "posta"
PRINT INSTR(10, A1$, A2$)          <F5>
  0
```

dato che la stringa "posta" non compare dopo la decima lettera di "capostazione". Questa possibilità è utile per trovare *ogni* ricorrenza di *a2$* all'interno di *a1$,* e non solamente la *prima,* come vedremo in un esempio nel paragrafo "Cicli DO...LOOP" (→ pag. 71)

LCASE$, UCASE$

Può essere talvolta necessario cambiare le lettere maiuscole in minuscole, o viceversa, come nella ricerca di una particolare successione di caratteri in un file lungo; ad esempio, si può volere che le parole FINE, fine e Fine siano considerate le stesse. Ciò si ottiene con le funzioni LCASE$ e UCASE$, che eseguono le seguenti conversioni:

- LCASES fornisce una copia della stringa suo argomento, nella quale tutte le lettere maiuscole sono convertite in minuscole.
- UCASE$ fornisce una copia della stringa suo argomento, nella quale tutte le lettere minuscole sono convertite in maiuscole. Esempio:

```
Tit$ = "Tabella ASCII"
PRINT Tit$
PRINT LCASE$(Tit$)
PRINT UCASE$(Tit$)                 <F5>
  Tabella ASCII
  tabella ascii
  TABELLA ASCII
```

LEFT$, RTRIM$

Queste due funzioni forniscono i caratteri di una stringa a partire dalla sua sinistra: in particolare la funzione

```
LEFT$(a$, n)
```

fornisce gli *n* caratteri più a sinistra della stringa *a$,* mentre la funzione

```
RTRIM$(a$)
```

fornisce la parte sinistra della stringa *a$,* dopo averne eliminato gli eventuali spazi finali. Esempio:

```
A$ = "ferrovia     "
PRINT LEFT$(A$, 5)
PRINT RTRIM$(A$)                   <F5>
  ferro
  ferrovia
```

La funzione RTRIM$ è utile per confrontare stringhe di lunghezza variabile e fissa: se ad esempio si scrivono le seguenti istruzioni

```
DIM Strinfis AS STRING * 10
Strinfis = "Si"
Strinvar = "Si"
```

le due stringhe Strinfis e Strinvar risultano diverse (dato che la prima ha 10 caratteri e la seconda 2), mentre RTRIM$(Strinfis) risulta uguale a Strinvar.
Le stesse operazioni di LEFT$ e RTRIM$, ma a partire dalla destra, sono svolte dalle funzioni RIGHT$ e LTRIM$.

LEN (*a$*)
Fornisce il numero di caratteri della stringa *a$*. Esempio:

```
PRINT LEN ("VIA")                <F5>
  3
```

MID$
Permette di estrarre un numero qualsiasi di caratteri da un punto qualsiasi di una stringa. La forma sintattica è:

```
MID$ (a$, inizio, n)
```

dove *a$* è la stringa di partenza, *inizio* è la posizione del carattere da cui inizia l'estrazione, *n* è il numero di caratteri estratti. Esempio:

```
PRINT MID$ ("Capostazione", 3, 5) <F5>
 posta
```

RIGHT$, LTRIM$
Queste due funzioni forniscono i caratteri di una stringa a partire dalla sua destra; in particolare la funzione

```
RIGHT$ (a$, n)
```

fornisce gli *n* caratteri più a destra della stringa *a$,* mentre la funzione

```
LTRIM$ (a$)
```

fornisce la parte destra della stringa *a$,* dopo averne eliminato gli eventuali spazi iniziali. Esempio:

```
A$ = "ferrovia"
PRINT RIGHT$ (A$, 3)
PRINT LTRIMS (A$)                <F5>
via
ferrovia
```

STR$(*n*), VAL(*a$*)
QBASIC non consente di assegnare un'espressione di stringa a una variabile numerica, né un'espressione numerica a una variabile di stringa: infatti espressioni del tipo

```
1% = "1"
A$ = 7
```

produrrebbero un messaggio di errore.
Si possono però eseguire le necessarie conversioni usando le funzioni STR$ e VAL. Precisamente la funzione

```
STR$ (n)
```

trasforma il numero *n* in stringa, in modo che possa essere impiegato in qualsiasi operazione con le stringhe. Esempio:

```
A$ = STR$ (5)
PRINT "MI" + A$                    <F5>
MI  5
```

Osserviamo che STR$ inserisce lo spazio iniziale riservato da QBASIC per i numeri positivi (vedi paragrafo "Istruzione PRINT", a pag. 26); questo può essere eliminato usando l'istruzione PRINT USING (vedi paragrafo "Visualizzazione di dati formattati", a pag. 27) oppure la funzione LTRIM$, come nel seguente esempio:

```
A$ = STR$ (5)
PRINT "MI" + LTRIM$ (A$)           <F5>
MI5
```

Invece la funzione

```
VAL (a$)
```

trasforma la stringa *a$* in numero, esaminandola a partire dal primo carattere e continuando la conversione fino a che trova caratteri riconoscibili come numeri; se. *a$* inizia con un carattere non numerico, il valore fornito è 0. Esempio:

```
PRINT VAL ("MI20121"), VAL ("14.07.1789") <F5>
  0   14
```

Nota. QBASIC dispone di altre funzioni che eseguono conversioni di numeri in stringhe (CV) e viceversa (MK), ereditate dalle versioni precedenti del BASIC, che le richiedevano per operazioni con file ad accesso casuale (→ pag. 135). Esse sono illustrate nel paragrafo "Aggiunta di dati nelle precedenti versioni di BASIC" (→ pag. 140), e tuttavia non sono più necessarie se nella definizione del record di un file ad accesso casuale si utilizza l'istruzione TYPE...END TYPE (illustrata al paragrafo "Definizione del campo del record", a pag. 137).

FUNZIONI VARIE

DATE$, TIME$
Sono due funzioni prive di argomento, che forniscono rispettivamente la data e l'ora corrente del sistema, sotto forma di stringa. Esempio:

```
PRINT DATE$
PRINT TIME$          <F5>
 02-07-1994
 09:12:45
```

Se si deve cambiare la data del sistema, si usa l'omonima istruzione DATE$, secondo il formato dell'esempio seguente:

```
DATE$ = "01-01-90"
```

Se si deve cambiare l'ora del sistema, si usa l'omonima istruzione TIME$, secondo uno dei formati degli esempi seguenti:

```
TIME$ = "11"
TIME$ = "11:24"
TIME$ = "11:24:39"
```

(se i minuti o i secondi non sono impostati, vengono posti automaticamente uguali a zero).
La data e l'ora del sistema sono aggiornate continuamente durante le sessioni di lavoro e, negli attuali personal computer, anche a macchina spenta, grazie a una batteria interna che alimenta l'orologio o *clock* di sistema.

TIMER
È una funzione priva di argomento, che fornisce il numero di secondi trascorsi dalla mezzanotte. Esempio:

```
PRINT TIME$
PRINT TIMER          <F5>
 10:00:00
 36000
```

Può essere usata per cronometrare l'esecuzione di una parte o di un interno programma, oppure come argomento dell'istruzione RANDOMIZE per generare un numero casuale (→ pag. 39).

IMMISSIONE DI DATI DALLA TASTIERA

La tastiera costituisce il principale dispositivo d'immissione di dati in un computer (altri dispositivi sono il mouse, lo scanner o un file di dati). QBASIC possiede due istruzioni - INPUT, LINE INPUT - e due funzioni - INPUT$, INKEY$ - che permettono di immettere dati da tastiera, e quindi assegnarli come contenuto a delle variabili (mentre le istruzioni LET e CONST viste al paragrafo "Assegnazione di valori da programma", (→ pag. 19), e le istruzioni READ...DATA che vedremo nel prossimo paragrafo "Lettura e assegnazione di valori", (→ pag. 98) eseguono la stessa assegnazione da programma).

INPUT

Una prima istruzione che permette di raccogliere i dati digitati sulla tastiera e assegnarli a una o più variabili è

```
INPUT [;] [messaggio;] variabile [, variabile] ...
```

dove il *messaggio* è una costante di stringa che informa l'utente sul tipo di dato da immettere. Come esempio di funzionamento dell'istruzione INPUT si può eseguire il seguente programma:

```
INPUT "Come ti chiami"; Nome$
INPUT "Quanti anni hai"; Anni
PRINT "Ciao "; Nome$; ", tu hai circa"; Anni * 365; "giorni"
```

Esso visualizza il *messaggio* "Come ti chiami?" e sospende l'esecuzione fino a che l'utente batta dei tasti seguiti da <Invio>; questo dato viene assegnato alla variabile Nome$. Compare poi il messaggio "Quanti anni hai" e quindi un messaggio che comunica all'utente la sua età espressa in giorni.
I dati che l'utente immette devono essere dello stesso tipo e numero delle variabili presenti nell'istruzione INPUT, altrimenti viene visualizzato un messaggio di errore. Per questa ragione è opportuno inserire in ogni caso nell'istruzione INPUT una *variabile* di stringa, ed eventualmente convertire il suo valore in numero, scrivendo ad esempio:

```
INPUT "Quanti anni hai"; Anni$: Anni = VAL(Anni$)
```

Se in una singola istruzione INPUT compaiono più variabili, l'utente deve immettere i loro valori separandoli con "," (e terminando al solito con <Invio>). Per questa ragione non è possibile immettere una stringa che contenga delle virgole (che verrebbero interpretate come separatori fra i dati immessi), mentre si possono immettere virgole e spazi vuoti solo racchiudendoli tra virgolette.
Se dopo la parola INPUT è inserito un ";" il cursore non avanza a linea nuova dopo l'immissione dei dati da parte dell'utente; quindi il successivo messaggio visualizzato compare sulla stessa linea. Si può provare eseguendo le due linee seguenti:

```
INPUT ; "Nome", N$
INPUT " Cognome", C$
```

Osserviamo che il ";" prima della *variabile* fa comparire un "?" dopo il *messaggio;* se il ";" è sostituito da una "," il "?" è soppresso.
È possibile dare un aspetto più "professionale" alla richiesta di dati che compare sul

video facendo seguire al *messaggio* una barra luminosa, che indica la lunghezza massima consentita per la *variabile*. Il programma che segue visualizza dopo la richiesta "Nome" una barra lunga 10 caratteri.

```
CLS
LOCATE 2, 1
PRINT "Nome: " + STRING$(10, 219)
LOCATE 2, 7
INPUT "", Nome$
LOCATE 2, 7 + LEN(Nome$)
PRINT SPACE$(10 - LEN(Nome$))
```

LINE INPUT

Se un programma deve assegnare a una variabile linee di testo che comprendono virgole o spazi vuoti iniziali o finali, e si vuole evitare all'utente il compito di racchiuderli tra virgolette, si usa l'istruzione LINE INPUT. Essa non visualizza un "?", e accetta in ingresso tutti i caratteri che si battono fino alla pressione di <Invio>. La forma sintattica è

```
LINE INPUT [;] [messaggio;] variabile
```

INPUT$(n)

A differenza delle istruzioni INPUT e LINE INPUT, che assegnano a una variabile il dato immesso dall'utente quando questi preme il tasto <Invio>, la funzione INPUT$(*n*) legge *n* caratteri battuti sulla tastiera (compresi i tasti di controllo, quali <Esc>, <BkSp> o <Invio>), e li assegna a una variabile. Ad esempio, se in risposta al seguente programma:

```
Prova$ = INPUT$(3)
PRINT Prova$
```

si batte tre volte il tasto <Esc>, lo schermo visualizza

←←←

cioè tre volte il simbolo associato al codice ASCII 27 del tasto <Esc>.

INKEY$

Quando QBASIC incontra un'espressione che contiene la funzione INKEY$, controlla se l'utente ha digitato un tasto a partire da:

- l'ultima volta che è stata usata un'espressione contenente INKEY$, oppure
- l'inizio del programma, se INKEY$ è usata per la prima volta.

Se in questo periodo non è stato digitato alcun tasto, INKEY$ fornisce una stringa vuota (""), altrimenti una stringa di 1 byte costituita dal carattere digitato.

A differenza di INPUT, INKEY$ non visualizza sullo schermo né un "?" né il carattere digitato; inoltre quest'ultimo è assegnato alla stringa appena si preme il tasto, senza bisogno che si confermi con <Invio>. Ciò velocizza l'uso dei programmi nei quali l'utente deve digitare un carattere in risposta a un messaggio sullo schermo, ma richiede che INKEY$ sia inserito all'interno di un ciclo, che sospende l'esecuzione del programma fino a che l'utente prema un tasto.

A questo proposito le due tecniche più diffuse sono rappresentate dai frammenti di programma seguenti:

```
PRINT "Premere un tasto qualsiasi per continuare"
10 A$ = INKEY$: IF A$ = "" THEN 10
PRINT "Fine"
```

che QBASIC mantiene dalle precedenti versioni (e infatti richiede un numero di linea nella seconda istruzione), e:

```
PRINT "Premere un tasto qualsiasi per continuare"
DO
LOOP UNTIL INKEY$ <> ""
PRINT "Fine"
```

che invece impiega la nuova struttura DO...LOOP, che sarà discussa nel successivo paragrafo "Cicli DO...LOOP" (→ pag. 71).

È anche possibile far ripartire il programma premendo un tasto specificato, scrivendo, ad esempio:

```
PRINT "Premere <ESC> per continuare"
DO
LOOP UNTIL INKEY$ = CHR$(27)
```

A differenza delle altre parole chiave discusse in questo paragrafo, INKEY$ permette che il programma esegua altre elaborazioni mentre è in attesa dell'ingresso da tastiera, come si può verificare inserendo, tra le istruzioni DO e LOOP scritte in precedenza, una coppia di istruzioni del tipo:

```
PRINT K
K = K + 1
```

Un esempio di utilizzo di alcune delle funzioni illustrate nei paragrafi precedenti è fornito dal seguente programma CALCOLATEMPO, che può essere impiegato per calcolare il tempo impiegato dal computer per compiere una determinata elaborazione. Il programma chiede all'utente, tramite l'istruzione INPUT, di scrivere l'ora iniziale, che viene assegnata alla variabile T1$ e l'ora finale, assegnata a T2$.
Quindi trasforma queste due stringhe in numeri decimali, per poterne eseguire la differenza, e di nuovo in una stringa nel formato hh/mm/ss.
Per controllare di avere scritto esattamente il programma potete provare a immettere dei valori di prova e confrontare la risposta fornita dal computer con quella attesa.
Questa è anzi una pratica che andrebbe seguita ogni volta che si realizza un nuovo programma.

```
REM ***CALCOLATEMPO***
CLS
10 INPUT "Ora iniziale (hh/mm/ss)"; T1$
INPUT "Ora finale (hh/mm/ss)"; T2$
IF T2$ < T1$ THEN PRINT "Riprova!": GOTO 10
H1 = VAL(LEFT$(T1$, 2))
H2 = VAL(LEFT$(T2$, 2))
M1 = VAL(MID$(T1$, 4, 2))
M2 = VAL(MID$(T2$, 4, 2))
S1 = VAL(RIGHT$(T1$, 2))
S2 = VAL(RIGHT$(T2$, 2))
T1 = H1 * 3600 + M1 * 60 + S1
T2 = H2 * 3600 + M2 * 60 + S2
T = T2 - T1
H = INT(T / 3600)
RS = T - 3600 * H
M = INT(RS / 60)
S = RS - 60 * M
PRINT "Il tempo impiegato e'"; H; " ore,"; M;
" minuti,"; S; " secondi."
```

IL PROGRAMMA: PRIMI CONCETTI

I diversi esempi di istruzioni viste finora avevano la caratteristica comune di far eseguire al computer una serie di operazioni secondo un ordine rigorosamente sequenziale. Questo modo di operare sfrutta però solo una minima parte delle potenzialità di un computer, per cui si può dire che chi si limitasse a usare QBASIC per scrivere comandi del tipo PRINT EXP(2) o PRINT 10 MOD 3, userebbe certamente uno strumento sovradimensionato rispetto alle proprie esigenze.
Infatti la potenza e la versatilità di un computer si manifestano nell'esecuzione ripetuta - anche centinaia o migliaia di volte - di *cicli* di operazioni (argomento che vedremo nel capitolo "Strutture cicliche", a pag. 61), oppure nella possibilità di fargli prendere una decisione sulle istruzioni da eseguire (nel capitolo "Strutture di decisione", a pag. 79), in base ai risultati di operazioni eseguite in precedenza. Queste prestazioni si ottengono scrivendo un **programma**, cioè un elenco ordinato e organizzato di comandi che descrivono il procedimento elaborativo (o *algoritmo*) desiderato, in una forma comprensibile al computer. I comandi di un programma sono detti anche *istruzioni,* e possono essere preceduti da un numero intero o da una parola detto *etichetta di riga;* l'etichetta è utile se nel corso del programma si dovrà fare riferimento a una determinata riga di istruzioni. Una riga di programma può contenere più istruzioni, separate con due punti. Ad esempio, è equivalente scrivere:

```
LOCATE 12, 40
PRINT "Centro dello schermo"
```

oppure:

```
LOCATE 12, 40: PRINT "Centro dello schermo"
```

COMMENTI IN UN PROGRAMMA (REM)

È spesso utile inserire in un programma delle linee di commento, per dargli un titolo o per far capire a chi legge il listato il tipo di operazione eseguito in quel punto. A questo fine basta far precedere il testo del commento dalla parola chiave

```
REM
```

che ha l'effetto di far ignorare a QBASIC tutti i caratteri che seguono sulla sua riga (si dice anche che REM è un'istruzione *non eseguibile*).
I commenti si possono anche inserire dopo le istruzioni di una riga di programma, facendoli precedere da un apice (').
È comunque buona pratica non eccedere nell'uso dei commenti, per non sprecare inutilmente spazio di memoria.

FINE DI UN PROGRAMMA (END)

Se in un programma è presente l'istruzione

```
END
```

questa è l'ultima eseguita, e le eventuali istruzioni che seguono sono ignorate. L'istruzione END può essere impiegata per far terminare l'esecuzione di un programma al verificarsi di un determinato evento, come nel caso dell'istruzione

```
IF    THEN END
```

che vedremo nel capitolo "Strutture di decisione" (→ pag. 79).



Capitolo 8

Strutture cicliche



CHE COS'È UN CICLO

La logica di un programma, in assenza di istruzioni che ne controllino il flusso, procede attraverso le sue istruzioni da sinistra a destra, o dall'alto al basso. Anche se è possibile scrivere programmi molto semplici con questo flusso unidirezionale, la maggior parte della potenza e utilità dei linguaggi di programmazione deriva dalla loro capacità di cambiare l'ordine di esecuzione delle istruzioni con strutture cicliche e con strutture di decisione (che vedremo nel prossimo capitolo).
Le **strutture cicliche** fanno eseguire ripetutamente a un programma un blocco di istruzioni (il *ciclo*), o per un determinato numero di volte, o fino a che una certa condizione (la *condizione di ciclo*) risulti vera o falsa. Vedremo dapprima le strutture FOR...NEXT e WHILE...WEND, già presenti nelle precedenti versioni del BASIC, quindi la struttura DO...LOOP, che QBASIC ha ripreso da altri linguaggi di programmazione.

CICLO FOR...NEXT

Il ciclo FOR...NEXT ripete le istruzioni contenute al suo interno per un numero determinato di volte, contando da un valore iniziale a uno finale aumentando o diminuendo il contenuto di una variabile detta *contatore di ciclo*. Il ciclo continua la sua esecuzione fino a che il *contatore* non ha raggiunto il valore finale.
La sintassi dell'istruzione è:

```
FOR contatore = inizio TO fine [STEP passo]
.
.
NEXT [contatore]
```

dove: *contatore* è una variabile numerica, che assume inizialmente il valore dell'espressione *inizio;* alla fine di ogni ciclo, l'istruzione NEXT incrementa o decrementa il valore del *contatore* del *passo* indicato (che può essere un numero intero positivo o negativo); se non è presente l'opzione STEP, il valore di *passo* è posto automaticamente uguale a 1. Il valore del *contatore* è quindi confrontato con *fine* e il ciclo ha termine se si verifica una di queste due circostanze:

- il ciclo conta in avanti (il passo è positivo) e il *contatore* è maggiore di *fine*
- il ciclo conta all'indietro (il passo è negativo) e il *contatore* è minore di *fine*.

Ad esempio, il programma

```
FOR I% = 1 TO 6
 PRINT I%, I%*I%
NEXT I%
```

produce la seguente uscita

```
1        1
2        4
3        9
4       16
5       25
6       36
```

La Figura 11 illustra la logica del ciclo FOR...NEXT quando il valore del *passo* è positivo, la Figura 12 quando il valore del *passo* è negativo.

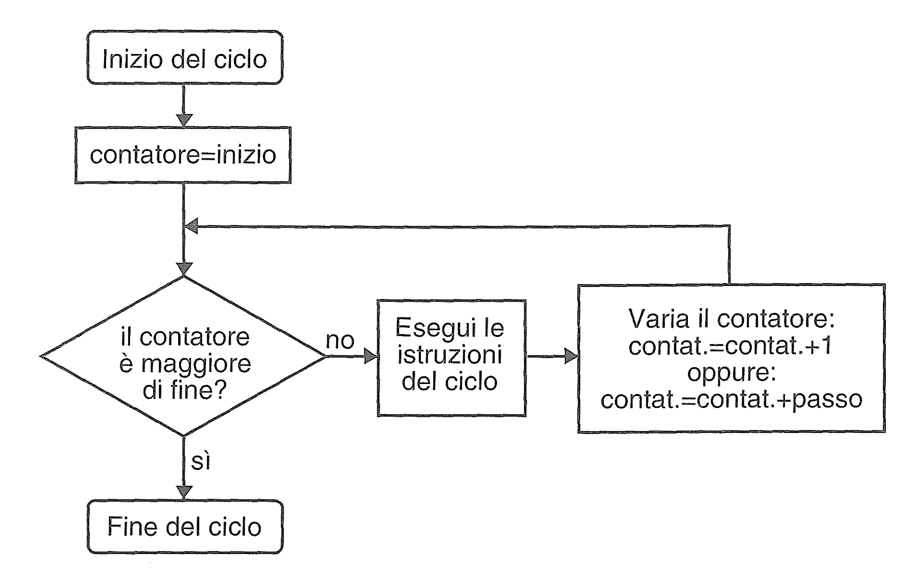


***Figura 11** - Logica del ciclo FOR...NEXT con* passo *positivo*

Le istruzioni FOR...NEXT eseguono sempre un test all'inizio, cosicché il ciclo non viene eseguito se si verifica una delle seguenti condizioni:

- il *passo* è positivo e il valore di *inizio* è maggiore di quello di *fine*. Esempio:

```
FOR I% = 10 TO 9
    .
    .
NEXT I%
```

- il *passo* è negativo e il valore di *inizio* è minore di quello di *fine*. Esempio:

```
FOR I% = -10 TO -9 STEP -1
.
.
NEXT i%
```

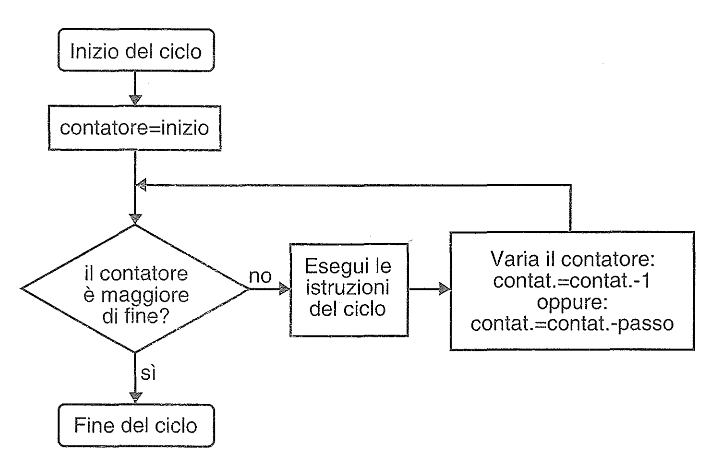


Figura 12 - Logica del ciclo FOR...NEXT con passo *negativo*

Annidamento di cicli

Un programma QBASIC può contenere più cicli FOR...NEXT l'uno dentro l'altro **(annidamento di cicli)**, che tuttavia non si devono "intersecare". Quindi si dovrà chiudere per primo l'ultimo ciclo che è stato aperto, poi il penultimo e così via. In altre parole, il seguente annidamento è corretto

```
FOR I% = 1 TO 10
 FOR J% = -5 TO 0
 .
 .
 NEXT J%
NEXT I%
```

mentre il seguente non è corretto

```
FOR I% = 1 TO 10
 FOR J% = -5 TO 0
 .
 .
 NEXT I%
NEXT J%
```

Se tutti i cicli annidati hanno lo stesso punto finale, possono essere chiusi da un'unica clausola NEXT. Questa dovrà contenere l'elenco dei contatori dei cicli a partire da quello più interno a quello più esterno. Pertanto i due programmi

```
FOR I% = 1 TO 2
 FOR J% = 4 TO 5
  PRINT I%, J%
 NEXT J%
NEXT I%
```

e

```
FOR I% = 1 TO 2
 FOR J% = 4 TO 5
  PRINT I%, J%
NEXT J%, I%
```

produrranno la stessa uscita, e cioè

```
     1          4
     1          5
     2          4
     2          5
```

Naturalmente, se si usa una clausola NEXT distinta per terminare ciascun ciclo, il numero delle clausole NEXT deve essere uguale a quello delle clausole FOR.
Nella clausola NEXT non è obbligatorio indicare il *contatore,* tuttavia, nel caso in cui si chiudano più cicli annidati con altrettante clausole NEXT, l'indicazione del *contatore* può essere di aiuto per capire in quale ciclo ci si trovi.

Uscita da un ciclo FOR...NEXT con EXIT FOR

Talvolta può essere necessario uscire da un ciclo FOR...NEXT prima che il contatore raggiunga il suo valore finale. Ciò si ottiene con l'istruzione EXIT FOR, come avviene nel seguente programma

```
FOR I% = 1 TO 30000
 PRINT SQR(I%)
 IF INKEY$ <> "" THEN EXIT FOR
NEXT I%
```

Esso visualizza le radici quadrate dei primi 30000 interi, e si ferma quando si preme un tasto qualsiasi.
Un ciclo FOR...NEXT può avere un numero qualsiasi di istruzioni EXIT FOR, situate in punti qualsiasi.
EXIT FOR ha effetto solo per il più piccolo ciclo FOR...NEXT in cui si trova. Ad esempio, se si preme un tasto mentre vengono eseguiti i seguenti cicli annidati

```
FOR I% = 1 TO 100
 FOR J% = 1 TO 100
  PRINT I%/J%
  IF INKEY$ <> "" THEN EXIT FOR
 NEXT J%
NEXT I%
```

il programma esce dal ciclo interno, ma continua a eseguire quello esterno, finché il contatore i% non ha raggiunto il valore finale.

PAUSA IN UN PROGRAMMA CON FOR...NEXT

Per inserire una pausa nell'esecuzione di un programma si può usare un ciclo "vuoto" FOR...NEXT del seguente tipo

```
FOR I% = 1 TO 10000: NEXT I%
```

Questo metodo va bene per pause brevi, o che comunque non debbano avere una durata esatta. Tuttavia differenti computer, differenti versioni di BASIC o differenti opzioni di compilazione possono far variare ampiamente la durata della pausa. Una tecnica migliore si basa sul ciclo DO...LOOP, come vedremo nel paragrafo "Pausa in un programma con DO... LOOP" (→ pag. 122).

ESPRESSIONI BOOLEANE

Le altre strutture cicliche di QBASIC (WHILE...WEND e DO...LOOP) nonché le strutture di decisione utilizzano le **espressioni booleane** (o logiche), cioè delle espressioni che possono essere o "vere" o "false", e che quindi è opportuno esaminare a questo punto. Nella sua forma più semplice, un'espressione booleana è costituita da due variabili raccordate da uno degli operatori di relazione già visti al paragrafo "Confronto di stringhe" (→ pag. 48). Ad esempio, X > Y è un'espressione booleana se alle variabili X e Y sono stati assegnati in precedenza dei valori, cosicché si possa dire se essa è vera o falsa. In un'espressione booleana possono comparire anche due variabili di stringa: in tal caso "minore" e "maggiore" si riferiscono all'ordine alfabetico, nel senso di "precede" e "segue". Così, ad esempio, l'espressione "alto" < "basso" è vera, perché la parola "alto" precede in ordine alfabetico la parola "basso".

OPERATORI BOOLEANI

Due o più espressioni booleane del tipo visto si possono combinare tra loro per mezzo degli **operatori booleani** (o logici)

AND OR XOR IMP EQV

per formare espressioni booleane più complesse. Questi operatori sono caratterizzati dal tipo di risultato che forniscono (vero o falso) quando collegano due espressioni booleane (a loro volta vere o false). I loro nomi e comportamenti sono riassunti in Tabella 11.

operatore	nome	produce un'espressione vera se collega due espressioni...
AND	congiunzione	entrambe vere
OR	disgiunzione	delle quali almeno una sia vera
XOR	OR esclusivo	delle quali solo una sia vera
IMP	implicazione	entrambe vere, o entrambe false, o falsa la 1ª e vera la 2ª
EQV	equivalenza	entrambe vere o entrambe false

***Tabella 11** - Operatori booleani usati in QBASIC*

Vediamo alcuni esempi.

L'espressione

```
(X >= 0) AND (X <= 5)
```

è vera se X è compreso tra 0 e 5, estremi inclusi.

L'espressione

```
(A < 0) OR (A > 0)
```

è vera se A è diverso da 0.

L'espressione

```
(B >= 0) XOR (B > 5)
```

è vera se B è compreso fra 0 e 5, estremi inclusi (in quanto allora risulta B >= 0 ma non B > 5).

Le espressioni

```
(5 = SQR(25)) IMP (3 = SQR(9))
(4 = SQR(25)) IMP (5 = SQR(9))
(4 = SQR(25)) IMP (3 = SQR(9))
```

sono tutte vere.

Le espressioni

```
("rosso" > "rossa") EQV (10 > 5)
(10/2 = 4) EQV ("alfa" > "beta")
```

sono entrambe vere.

Negli esempi precedenti le parentesi non sono in realtà necessarie, in quanto QBASIC valuta gli operatori di relazione *prima* degli operatori logici; esse tuttavia aiutano a rendere più leggibile un'espressione booleana complessa.

Nota. QBASIC assegna i valori numerici -1 e 0 alle espressioni booleane vere e false, cioè:

FALSO $\leftrightarrow$ 0
VERO $\leftrightarrow$ -1

come si può verificare con i semplici comandi mostrati in Tabella 12.

il comando...	produce l'uscita...
PRINT 10/2 = 5	-1
PRINT 5 > 10	0

Tabella 12 *- Valori numerici di espressioni booleane*

Esiste infine l'operatore logico

NOT

che applicato a un'espressione booleana ne muta il valore da "vero" in "falso" e viceversa.

Dal punto di vista tecnico, NOT inverte ogni bit della rappresentazione binaria del suo operando, cioè cambia gli "0" in "1" e gli "1" in "0". Perciò, dato che il valore 0 (corrispondente a falso) è memorizzato internamente come sequenza di sedici bit 0, NOT 0 (corrispondente a vero) è memorizzato internamente come sequenza di sedici bit 1; cioè, riassumendo,

FALSO	↔	0000 0000 0000 0000
VERO (=NOT FALSO)	↔	1111 1111 1111 1111

Nel metodo del complemento a 2, che QBASIC usa per memorizzare gli interi, sedici bit 1 rappresentano il valore -1.

Nota. QBASIC fornisce il valore -1 quando valuta un'espressione booleana vera, e considera vero ogni valore diverso da zero, come risulta dal seguente esempio.

il programma...	produce l'uscita...
INPUT "Scrivi un numero"	Scrivi un numero : 2
IF X THEN PRINT X; " è vero"	2 è vero

CICLO WHILE...WEND

Il ciclo FOR...NEXT è utile quando si conosce a priori il numero esatto di volte che l'iterazione va ripetuta. Se invece non si può prevedere tale numero, ma si conosce la condizione che dovrà porre fine al ciclo, si usa il ciclo WHILE...WEND. Esso infatti ripete le istruzioni contenute al suo interno fino a che la *condizione* di ciclo si mantiene vera, anziché in base al valore di un contatore.
La sintassi dell'istruzione è

```
WHILE condizione
.
.
WEND
```

dove *condizione* è una qualsiasi espressione booleana (→ pag. 67), la cui **verità** determina l'esecuzione del ciclo.
Ad esempio, il programma

```
INPUT R$
WHILE R$ <> "S" AND R$ <> "N"
 PRINT "La risposta deve essere S o N"
 INPUT R$
WEND
```

richiede di immettere una risposta consistente in una "S" o una "N", e visualizza il messaggio indicato nella riga PRINT se si immette una risposta diversa.
La logica del ciclo WHILE...WEND è illustrata in Figura 13

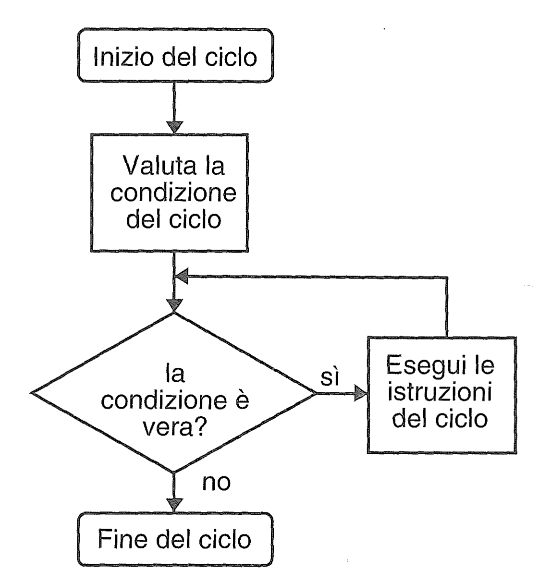


***Figura 13** - Logica dei cicli WHILE...WEND e DO WHILE...LOOP*

CICLI DO...LOOP

In maniera analoga al ciclo WHILE...WEND, anche il ciclo DO...LOOP esegue un blocco di istruzioni un numero di volte non stabilito a priori, dato che l'uscita dal ciclo dipende dalla verità di una *condizione.*
Tuttavia, a differenza di WHILE...WEND, il ciclo DO...LOOP permette di valutare sia la **verità** sia la **falsità** della *condizione,* e di porre la verifica sia all'inizio che alla fine del ciclo; esso possiede quindi quattro forme sintattiche.

La **prima forma sintattica**, con verifica all'inizio del ciclo, è:

```
DO [WHILE condizione]

  [⋮]

 [EXIT DO]
LOOP
```

dove *condizione* è una qualsiasi espressione logica, la cui **verità** determina l'esecuzione del ciclo. La logica è pertanto la stessa del ciclo WHILE...WEND (già illustrata in Figura 13).
Un esempio è costituito dal seguente programma

```
INPUT "Scrivi un carattere e avrai il suo codice ASCII", car$
DO WHILE (car$ <> "")
 PRINT "Il codice ASCII di "; car$; " è"; ASC(car$)
 INPUT "Scrivi un'altra lettera (o <Invio> per finire)", car$
LOOP
```

Esso chiede di scrivere una lettera e ne fornisce il codice ASCII (→ pag. 14); il programma ha termine quando si preme il tasto <Invio>.

La **seconda forma sintattica**, con verifica all'inizio del ciclo, è:

```
DO [UNTIL condizione]

  [⋮]

 [EXIT DO]
LOOP
```

dove *condizione* è una qualsiasi espressione logica, la cui **falsità** determina l'esecuzione del ciclo (vedi Figura 14).

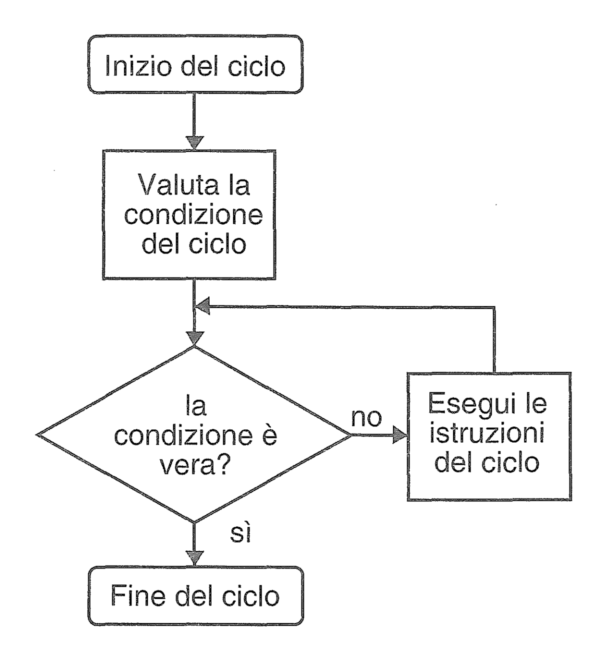


Figura 14 - Logica del ciclo DO UNTIL...LOOP

Un esempio è costituito dal seguente frammento di programma:

```
DO UNTIL INKEY$ <> ""
LOOP
```

Esso è un ciclo vuoto, che sospende l'esecuzione del programma in cui è inserito fino a che non venga premuto un tasto (cioè fino a che risulti falsa la condizione "è stato premuto un tasto").
È di solito preceduto da un'istruzione che visualizza sullo schermo un messaggio del tipo: "Premere un tasto qualsiasi per continuare".

La **terza forma sintattica**, con verifica alla fine del ciclo, è:

```
DO
 [:]
 [EXIT DO]
LOOP [WHILE condizione]
```

dove *condizione* è una qualsiasi espressione logica, la cui **verità** determina la ripetizione del ciclo (vedi Figura 15).

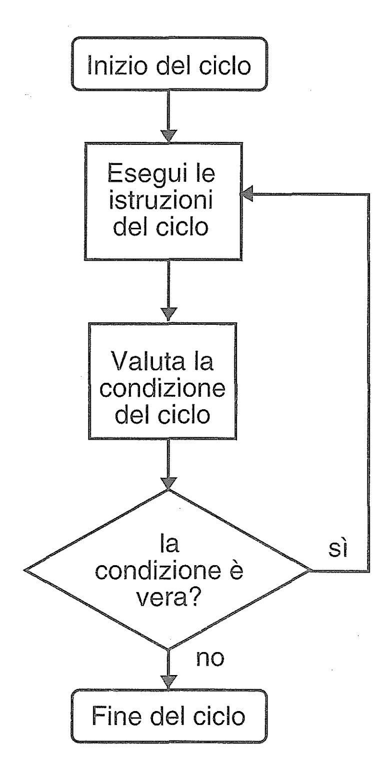


Figura 15 *- Logica del ciclo DO...LOOP WHILE*

Un primo esempio è costituito dal seguente programma:

```
RANDOMIZE TIMER
numcas% = INT(RND*100) + 1
DO
 INPUT "Che numero ho pensato"; numuten%
 IF (numuten% > numcas%) THEN
  PRINT "Troppo alto, riprova"
 END IF
 IF (numuten% < numcas%) THEN
  PRINT "Troppo basso, riprova"
 END IF
LOOP WHILE (numcas% <> numuten%)
```

In esso la prima istruzione imposta il generatore di numeri casuali (→ pag. 39) e la seconda genera un intero compreso fra 1 e 100; quindi l'istruzione INPUT chiede all'utente di indovinare il numero generato, mentre le due istruzioni IF...END IF indicano se ha fornito un valore troppo alto o troppo basso.

Un secondo esempio è costituito dal seguente programma, che utilizza la funzione INSTR (→ pag. 50) per trovare tutte le ricorrenze della parola “Satan” nel verso “Papè Satan, papè Satan, aleppe”.

```
S1$ = "Papè Satan, papè Satan, aleppe"
S2$ = "Satan"
PRINT S1$
Inizio = 1
Numric = 0
DO
 Ric = INSTR(Inizio, A1$, A2$)
 IF Ric > 0 THEN
  PRINT TAB(Ric) S2$
  Inizio = Ric + 1
  Numric = Numric + 1
 END IF
LOOP WHILE RIC
PRINT "Numero di ricorrenze ="; Numric
```

Esso produce la seguente uscita:

```
Papè Satan, papè Satan, aleppe
     Satan
                 Satan
Numero di ricorrenze = 2
```

La **quarta forma sintattica,** con verifica alla fine del ciclo, è:

```
DO
 [:]
[EXIT DO]
LOOP [UNTIL condizione]
```

dove *condizione* è una qualsiasi espressione logica, la cui **falsità** determina la ripetizione del ciclo (vedi Figura 16).

Un esempio è costituito dal seguente programma:

```
DO
 INPUT "Scrivi un numero (0 per finire)", num
LOOP UNTIL num = 0
PRINT "Fine del ciclo"
```

Esso chiede di scrivere un numero, e ripete la richiesta fino a che si scrive il valore 0, dopo di che il ciclo ha termine.

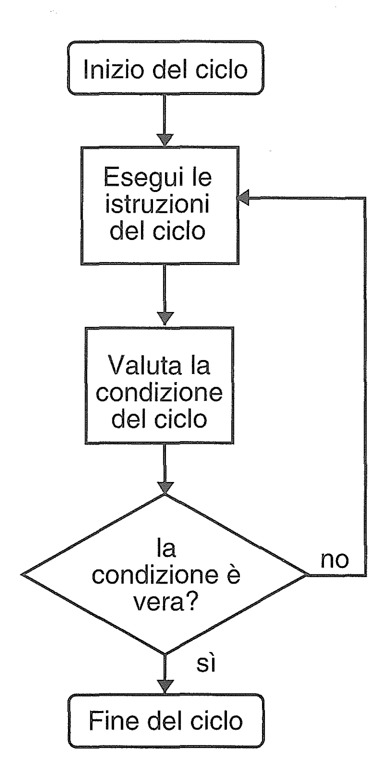


__Figura 16__ - Logica del ciclo DO...LOOP UNTIL

Naturalmente è sempre possibile riscrivere una *condizione* che contiene la parola WHILE con una che contiene UNTIL, come risulta dai due programmi seguenti, che producono lo stesso risultato:

```
DO WHILE NOT EOF(1)
 LINE INPUT #1, B$
 PRINT B$
LOOP

DO UNTIL EOF(1)
 LINE INPUT #1, B$
 PRINT B$
LOOP
```

Essi leggono una linea del file 1 e la stampano sullo schermo, e ciò fino alla fine del file, che viene riconosciuta quando la funzione EOF(1) fornisce valore vero (→ pag. 132).

CONFRONTO TRA DO...LOOP E WHILE...WEND

Se il controllo è posto alla fine del ciclo DO...LOOP, le istruzioni del ciclo vengono sempre eseguite almeno una volta, mentre con il ciclo WHILE...WEND può essere necessario ricorrere a un trucco per forzare la prima esecuzione del ciclo.
Ad esempio, per ripetere un gruppo di istruzioni se l'utente preme il tasto "S", con il ciclo WHILE...WEND è necessario assegnare il valore "S" a una variabile fuori dal ciclo:

```
risp$ = "S"
WHILE UCASE$(risp$) = "S"
 .
 .
 INPUT "Vuoi continuare"; risp$
WEND
```

mentre con il ciclo DO...LOOP ciò non è necessario:

```
DO
 .
 .
 INPUT "Vuoi continuare"; risp$
LOOP WHILE UCASE$(risp$) = "S"
```

USCITA DA UN CICLO DO...LOOP CON EXIT DO

Di solito all'interno di un ciclo DO...LOOP si verifica qualcosa che alla fine cambia la condizione di ciclo da vera in falsa o viceversa, ponendo fine al ciclo stesso. Negli esempi visti finora, la verifica si trovava all'inizio o alla fine del ciclo; tuttavia, se si usa l'istruzione EXIT DO per uscire dal ciclo, la verifica può essere messa all'interno del ciclo.
Un ciclo DO...LOOP può contenere un numero qualsiasi di istruzioni EXIT DO, che si possono trovare in punti qualsiasi.

Ad esempio, il seguente programma

```
INPUT "Sequenza da cercare: ", seqe$
OPEN "ELENCO" FOR INPUT AS #1
DO UNTIL EOF(1)
 LINE INPUT #1, tempor$
 IF INSTR(tempor$, seque$) > 0 THEN
  PRINT tempor$
  EXIT DO
 END IF
LOOP
```

apre il file ELENCO e lo legge una riga alla volta, fino a che si raggiunge la fine del ciclo oppure viene trovata una sequenza di caratteri indicata dall'utente. Se la sequenza viene trovata prima della fine del file, un'istruzione EXIT DO fa uscire dal ciclo (comunque su questo tipo di file torneremo al paragrafo "File sequenziali: lettura", a pag. 132).

MENU DI SCELTA CON DO...LOOP

L'istruzione DO...LOOP può essere usata anche senza indicare una *condizione,* né all'inizio né alla fine del ciclo. In tal caso il ciclo verrebbe eseguito all'infinito, per cui la condizione di uscita si deve trovare al suo interno. Un esempio tipico di questa situazione si ha nei **menu di scelta,** che sono realizzati da programmi simili al seguente.

```
DO
  CLS
  PRINT "Menu principale"
  PRINT
  PRINT "1  Aggiunta nuove schede"
  PRINT "2  Cancellazione schede"
  PRINT "3  Visualizzazione schede"
  PRINT "4  Uscita"
  PRINT
  PRINT "Effettua una scelta (1-4)"
  A$ = INKEY$
  SELECT CASE A$
    CASE "1"
      CALL AggSchede
    CASE "2"
      CALL CancSchede
    CASE "3"
      CALL VisSchede
    CASE "4"
      EXIT DO
    CASE ELSE
      BEEP
  END SELECT
LOOP
END
```

Esso utilizza l'istruzione SELECT CASE, che vedremo nel prossimo capitolo, per far eseguire una tra più azioni possibili (Aggiunta nuove schede, Cancellazione schede, Visualizzazione schede) da altrettanti sottoprogrammi (AggSchede, CancSchede, VisSchede), oppure l'Uscita dal ciclo. I sottoprogrammi sono richiamati dall'istruzione CALL, della quale parleremo più in dettaglio nel paragrafo "Sottoprogrammi" (→ pag. 111), a proposito della programmazione modulare.

Capitolo 9

Strutture di decisione



PROGRAMMI FLESSIBILI

Quando incontra una **struttura di decisione**, un programma valuta un'espressione, quindi si dirama verso diversi blocchi di istruzioni a seconda del risultato della valutazione.
Le strutture di decisione disponibili in QBASIC sono:

- il blocco di istruzioni IF...THEN...[ELSE]
- l'istruzione SELECT CASE
- l'istruzione ON...GOSUB

Ciascuna di esse permette alla logica del programma di determinare l'aspetto del codice, anziché il numero di istruzioni che si possono incrociare in una linea. Ciò fornisce non solo il vantaggio di una maggiore flessibilità nel programmare, ma anche quello di una migliore leggibilità del programma e facilità di manutenzione, una volta terminato.
A seconda del valore di una condizione, le strutture di decisione fanno sì che un programma compia una delle due seguenti azioni:

- esecuzione di una o più istruzioni alternative, interne alla stessa struttura di decisione
- diramazione a un'altra parte del programma, esterna alla struttura di decisione.

IF...THEN...ELSE SU UNA SOLA LINEA

Nelle precedenti versioni del BASIC, l'istruzione IF...THEN...ELSE su una sola linea è l'unica che permette di prendere una decisione. Nella sua forma più semplice, che ha la seguente forma sintattica IF `condizione` THEN `istruzione`, l'istruzione IF...THEN valuta la *condizione* che segue la parola IF e, se la trova vera (cioè diversa da zero), esegue l'*istruzione* che segue la parola THEN; se la *condizione* è falsa (cioè uguale a zero), il programma continua con la linea successiva a quella dell'istruzione IF...THEN. Un esempio è fornito dal seguente frammento di programma, scritto in BASICA, (ma eseguibile anche in QBASIC) che fa svolgere tre operazioni diverse a seconda del valore che viene immesso da tastiera.

```
30 INPUT A
40 IF A > 100 THEN PRINT "Troppo grande" : GOTO 30
50 IF A = 100 THEN GOTO 70
60 PRINT A/100 : GOTO 30
70 END
```

Con l'aggiunta dell'opzione ELSE, si può far eseguire al programma un gruppo di azioni (quelle che seguono la parola THEN) se l'espressione è vera, e un altro gruppo di azioni (quelle che seguono la parola ELSE) se l'espressione è falsa.

Un esempio è fornito dal seguente segmento di programma, che controlla l'immissione della parola d'ordine corretta.

```
10 INPUT "Parola d'ordine"; PAR$
20 IF PAR$ = "chiave" THEN 50 ELSE PRINT "Riprova"
: GOTO 10
```

L'istruzione IF...THEN...ELSE su una sola linea, sebbene adeguata per decisioni semplici, dà luogo a un codice pressoché illeggibile se si devono scrivere decisioni complicate. Ciò è particolarmente vero se si scrive un programma nel quale tutte le azioni alternative siano indicate entro la stessa istruzione IF...THEN...ELSE, o se più istruzioni IF...THEN...ELSE vengono annidate (poste cioè l'una dentro l'altra), costruzione, peraltro, perfettamente legale. Come esempio della difficoltà di seguire un controllo anche molto semplice, consideriamo il seguente segmento di programma.

```
20 INPUT A,B,
30 IF A <=50 THEN IF B <= 50 THEN PRINT " A<= 50,
B <= 50" ELSE PRINT "A <= 50, B > 50" ELSE IF B <= 50
THEN PRINT "A > 50, B <= 50" ELSE PRINT "A > 50,
B > 50"
```

IF...THEN...ELSE NELLA FORMA A BLOCCHI

Per evitare istruzioni complicate come l'ultima del paragrafo precedente, QBASIC mette a disposizione l'istruzione IF...THEN...ELSE nella forma a blocchi, che non restringe più la decisione a una sola linea logica. Il precedente segmento di programma si può allora riscrivere nella seguente forma, molto più comprensibile:

```
INPUT A,B
IF A <= 50 THEN
  IF B <= 50 THEN
    PRINT "A <= 50, B <= 50"
  ELSE
    PRINT "A <= 50, B > 50"
  END IF
ELSE
  IF B <= 50 THEN
    PRINT "A > 50, B <= 50"
  ELSE
    PRINT "A > 50, B > 50"
  END IF
END IF
```

La sintassi dell'istruzione IF...THEN...ELSE nella forma a blocchi è

```
IF condizione1 THEN
  [blocco-1]
[ELSEIF condizione2 THEN
  [blocco-2]]

.
.
[ELSE
  [blocco-n]]
END IF
```

dove gli argomenti *condizione1, condizione2,...* sono espressioni o numeriche - nel qual caso ogni valore diverso da zero è considerato vero, mentre zero è falso - oppure booleane (→ pag. 67). Ogni clausola IF, ELSEIF ed ELSE è seguita da un *blocco* di istruzioni; nessuna istruzione del *blocco* si può trovare sulla stessa linea di una clausola IF, ELSEIF o ELSE, altrimenti QBASIC la considera un'istruzione IF...THEN...ELSE su una sola linea.
Come esempio consideriamo il seguente frammento di programma, che conta e dichiara numeri positivi e negativi

```
IF X > 0 THEN
  PRINT "X è positivo"
  NUMPOS = NUMPOS + 1
ELSEIF X < 0 THEN
  PRINT "X è negativo"
  NUMNEG = NUMNEG + 1
ELSE
  PRINT "X è zero"
END IF
```

QBASIC valuta ciascuna espressione presente nelle clausole IF ed ELSEIF dall'alto in basso, saltando i blocchi di istruzioni finché non trova la prima espressione vera; in tal caso esegue le istruzioni corrispondenti all'espressione, quindi esce dal blocco e va all'istruzione che segue la clausola END IF.
Se nessuna delle espressioni contenute nelle clausole IF o ELSEIF è vera, QBASIC salta alla clausola ELSE, se presente, ed esegue le sue istruzioni. Se non vi è una clausola ELSE, il programma continua con l'istruzione che segue la clausola END IF.
Le clausole ELSE ed ELSEIF sono entrambe opzionali, come si vede nel seguente esempio

```
IF X < 100 THEN
  PRINT X
  NUM = NUM + 1
END IF
INPUT "Nuovo valore"; RISP
```

Un singolo blocco IF ... THEN ... ELSE può contenere più di un'istruzione ELSEIF, come mostra il seguente frammento di programma, che identifica il tipo di un carattere comunicato al computer da tastiera o già presente in memoria

```
IF C$ >= "A" AND C$ <= "Z" THEN
  PRINT "Lettera maiuscola"
ELSEIF C$ >= "a" AND C$ <= "z" THEN
  PRINT "Lettera minuscola"
ELSEIF C$ >= "0" and C$ <= "9" THEN
  PRINT "Numero"
ELSE
  PRINT "Non alfanumerico"
END IF
```

I blocchi IF...THEN...ELSE possono anche essere nidificati, cioè se ne può inserire uno dentro un altro come nel seguente esempio, che identifica i segni di tre numeri comunicati o già noti al computer

```
IF X > 0 THEN
  IF Y > 0 THEN
    IF Z > 0 THEN
      PRINT "Tutti positivi"
    ELSE
      PRINT "Solo X e Y sono positivi"
    END IF
  END IF
ELSEIF X = 0 THEN
  IF Y = 0 THEN
    IF Z = 0 THEN
      PRINT "Tutti uguali a zero"
    ELSE
      PRINT "Solo X e Y uguali a zero"
    END IF
  END IF
ELSE
  PRINT "X negativo"
END IF
```

ISTRUZIONE SELECT CASE

L'istruzione SELECT CASE, che QBASIC ha ripreso da altri linguaggi di programmazione, è una struttura di decisione a scelta multipla simile all'istruzione IF...THEN...ELSE a blocchi, e può essere usata tutte le volte che si può usare IF...THEN...ELSE. Fra esse ci sono tuttavia alcune differenze, che vedremo nel prossimo paragrafo.
La sintassi dell'istruzione SELECT CASE è

```
SELECT CASE espressione
  CASE listal
  [blocco-1]
  [CASE lista2]
  [blocco-2]]
.
.
  [CASE ELSE
    [blocco-n]]
END SELECT
```

dove: l'*espressione* può essere una variabile sia numerica sia di stringa, e gli argomenti *listal, lista2,...* che seguono le clausole CASE possono essere uno o più dei seguenti, separati da virgole:

- un'espressione numerica o un intervallo di espressioni numeriche
- un'espressione di stringa o un intervallo di espressioni di stringa

Se il valore dell'*espressione* che segue SELECT CASE compare nella *lista* che segue una clausola CASE, viene eseguito il *blocco* di istruzioni con quella clausola CASE, quindi il programma passa a eseguire la prima istruzione che segue l'istruzione END SELECT. Perciò se alla richiesta del seguente programma (costituita da un "?" determinata dall'istruzione INPUT)

```
INPUT X
SELECT CASE X
  CASE 1
    Print "Uno"
  CASE 2
    Print "Due"
  CASE 3
    Print "Tre"
  END SELECT
PRINT "è tutto."
```

si risponde digitando da tastiera il numero "1", si ottiene la seguente uscita:

```
Uno, è tutto.
```

Se in un'istruzione CASE si vuole indicare un intervallo di espressioni, si usa una delle due forme sintattiche seguenti:

```
CASE espressione TO espressione
CASE IS espressione con operatore di relazione
```

dove l'*espressione* può essere sia di tipo numerico sia di stringa, e l'*operatore di relazione* è uno di quelli indicati in Tabella 10 di pag. 48.
Ad esempio, se si scrive

```
CASE 1 TO 4
```

le istruzioni associate sono eseguite quando l'*espressione* contenuta nell'istruzione SELECT CASE è maggiore o uguale a 1 e minore o uguale a 4. Se si scrive

```
CASE IS < 5
```

le istruzioni associate sono eseguite quando l'*espressione* è minore di 5.
Nel caso in cui si esprima un intervallo con la parola chiave TO, bisogna stare attenti a scrivere per primo il valore più piccolo.
Ad esempio, l'espressione

```
CASE -5 TO -1
```

è corretta, mentre l'espressione

```
CASE -1 TO -5
```

sarebbe errata, e le istruzioni associate non verrebbero mai eseguite.
Analogamente, un intervallo di espressioni di stringa deve contenere gli estremi in ordine alfabetico, come nell'istruzione:

```
CASE "alfa" TO "beta"
```

In una clausosa CASE si possono elencare anche più espressioni o intervalli separandoli con virgole, come mostrano i seguenti esempi:

```
CASE 1 TO 4, 7 TO 9, num1%
CASE IS = nome$, IS = "fine del file"
CASE IS < "alto", "alto" TO "basso"
```

Può succedere che più di una clausola CASE contenga lo stesso valore (o intervallo di valori). In tal caso sono eseguite le sole istruzioni associate con la prima clausola, come mostra il seguente programma, che si aspetta l'immissione di una stringa lunga al massimo 10 caratteri:

```
INPUT risp$
SELECT CASE risp$
 CASE "A" TO "AZZZZZZZZZ"
  PRINT "Parola in lettere maiuscole che inizia con A"
 CASE IS < "A"
  PRINT "Sequenza di caratteri non alfabetici"
 CASE "ABILE"
  PRINT "Caso particolare"
END SELECT
```

Se in risposta alla richiesta di questo programma (prodotta dall'istruzione INPUT) si immette la parola "ABILE", il programma risponde "Parola in lettere maiuscole che inizia con A", e non "Caso particolare". Infatti, l'istruzione CASE "ABILE" non viene mai eseguita, in quanto la parola "ABILE" rientra nell'intervallo della prima clausola CASE.

La clausola

```
CASE ELSE
```

se viene usata, deve essere l'ultima clausola CASE elencata nell'istruzione SELECT CASE. Le istruzioni eventualmente comprese fra CASE ELSE ed END SELECT sono eseguite solo se l'*espressione* non si identifica con alcuna delle altre alternative CASE dell'istruzione SELECT CASE.
Perciò è buona prassi inserire un'istruzione CASE ELSE in ogni blocco SELECT CASE, per gestire valori imprevisti dell'*espressione*. Se infatti l'*espressione* non si identifica con alcuna delle istruzioni CASE, viene visualizzato il messaggio

```
CASE ELSE expected
```

Questo messaggio comparirebbe, ad esempio, se l'utente avesse immesso la parola "BRAVO" in risposta alla richiesta del programma precedente.
Un blocco SELECT CASE deve terminare con l'istruzione

```
END SELECT
```

L'istruzione SELECT CASE viene usata spesso per visualizzare sullo schermo un *menu di scelta,* come abbiamo visto nel paragrafo Menu di scelta con DO...LOOP (→ pag. 77).
Quando il *blocco* di istruzioni risulta piuttosto lungo, è preferibile richiamarlo per mezzo dell'istruzione CALL, come vedremo nel successivo paragrafo "Sottoprogrammi" (→ pag. 111).

CONFRONTO TRA SELECT CASE E IF...THEN...ELSE

La differenza principale fra le istruzioni SELECT CASE e IF...THEN...ELSE è che SELECT CASE valuta una *singola* espressione, quindi esegue differenti istruzioni o fa diramare a differenti parti del programma a seconda del risultato. Invece un blocco IF...THEN...ELSE può valutare *diversi tipi* di espressioni.
Come esempio del diverso modo di operare delle istruzioni SELECT CASE e IF...THEN...ELSE, consideriamo i frammenti di programma riportati in Tabella 13, che esaminano il carattere immesso da tastiera, e determinano l'esecuzione di azioni diverse a seconda del carattere immesso.

Programma con SELECT CASE	Programma con IF...THEN...ELSE
INPUT X SELECT CASE X CASE 1 PRINT "Uno" CASE 2 PRINT "Due" CASE 3 PRINT "Rte" CASE ELSE PRINT "Il numero deve essere compreso fra 1 e 3" END SELECT	INPUT X IF X = 1 THEN PRINT "Uno" ELSEIF X = 2 THEN PRINT "Due" ELSEIF X = 3 THEN PRINT "Tre" ELSE PRINT "Il numero deve essere compreso fra 1 e 3"

Tabella 13 - Confronto tra SELECT CASE e IF...THEN...ELSE

L'istruzione IF...THEN...ELSE, essendo in grado di valutare diverse espressioni, può risultare più efficiente di SELECT CASE, come mostra il seguente esempio:

```
INPUT X, Y
  IF X = 0 AND Y = 0 THEN
    PRINT "entrambi zero"
  ELSEIF X = 0 THEN
    PRINT "solo X è zero"
  ELSEIF Y = 0 THEN
    PRINT "solo Y è zero"
  ELSE
    PRINT "nessuno è zero"
END IF
```

Capitolo 10

Vettori e matrici



CHE COSA SONO I VETTORI E LE MATRICI

Se un programma deve trattare insiemi piuttosto ampi di dati collegati tra loro, può risultare scomodo impiegare le variabili esaminate nel capitolo "Variabili e costanti" (→ pag. 17). Ad esempio, per richiedere l'immissione da tastiera dei nomi di 40 persone si dovrebbero scrivere altrettante istruzioni del tipo

```
INPUT "Scrivi il nome numero 1 "; Nome1$
.
.
INPUT "Scrivi il nome numero 40 "; Nome40$
```

La situazione sarebbe ancora più pesante se si dovessero eseguire su questi dati delle elaborazioni anche semplici, come ad esempio l'ordinamento alfabetico. È allora molto più comodo usare delle variabili con indici, dette a seconda dei casi vettori o matrici (in inglese *array*), al posto delle variabili semplici (o senza indici) finora considerate.
Precisamente, un **vettore** è un insieme di variabili, tutte con lo stesso nome seguito da un numero d'ordine progressivo racchiuso tra parentesi detto *indice*. Esempio:

```
Nome$(1)  Nome$(2) ... Nome$(40)
```

Le precedenti variabili si chiamano le *componenti* del vettore Nome$, e si dice anche che il vettore Nome$ è costituito dalle componenti Nome$(1), Nome$(2), ... Nome$(40).
Come risulta da questo esempio, il vettore ha un nome uguale a quello delle sue componenti, ma senza l'indicazione del numero d'ordine tra parentesi.
In certi casi può essere necessario usare un vettore con due indici o **matrice** a due dimensioni, come succederebbe se volessimo considerare le temperature relative alle diverse ore dei diversi giorni della settimana. In tal caso potremmo considerare la matrice Temp, le cui componenti potrebbero avere un primo indice relativo all'ora (e variabile da 0 a 23) detto indice di riga, e un secondo indice relativo al giorno (e variabile da 1 a 7) detto indice di colonna. La ragione di questa terminologia si spiega considerando lo schema di Tabella 14.

	lunedì	**martedì**	**mercoledì**	**...**	**domenica**
0	Temp(0, 1)	Temp(0, 2)	Temp(0, 3)		Temp(0, 7)
1	Temp(1, 1)	Temp(1, 2)	Temp(1, 3)		Temp(1, 7)
2	Temp(2, 1)	Temp(2, 2)	Temp(2, 3)		Temp(2, 7)
3	Temp(3, 1)	Temp(3, 2)	Temp(3, 3)		Temp(3, 7)
.					
.					
23	Temp(23, 1)	Temp(23, 2)	Temp(23, 3)		Temp(23, 7)

Tabella 14 *- Componenti di una matrice a due dimensioni*

Le componenti di un vettore o di una matrice sono variabili di uno dei tipi già visti al paragrafo "Tipi di variabili" (→ pag. 20), e cioè

Intero, Intero LONG, Singola precisione, Doppia precisione, Stringa

e naturalmente il loro contenuto deve essere coerente con il tipo cui appartengono.
In QBASIC si possono trattare anche matrici con più di due indici o. In realtà multidimensionale è una matrice con + di 1 dim.

DIMENSIONAMENTO (DIM, OPTION BASE)

Dato che i vettori e le matrici possono avere numerose componenti (il massimo valore consentito è 32767) e richiedere quindi grandi spazi di memoria, è necessario informare QBASIC se il numero di tali componenti supera un certo valore (11 elementi), *dimensionando* il vettore. Ciò si ottiene con una delle istruzioni

```
DIM vettore(n)[...]
DIM matrice(m, n)[...]
```

dove: i puntini indicano che si possono dimensionare con una stessa istruzione più vettori e/o matrici, separando i loro nomi con la virgola; *m* e *n* indicano i valori massimi che si intendono dare agli indici. Ad esempio, l'istruzione

```
DIM Nome$(40)
```

permette di usare per il vettore Nome$ un indice che assume al massimo il valore 40, mentre l'istruzione

```
DIM Temp(23, 7)
```

permette di usare per la matrice Temp un primo indice che assume al massimo il valore 23 e un secondo che può arrivare a 7. Se un vettore non ha più di 11 elementi, il dimensionamento non è necessario, ma è lo stesso opportuno, perché in sua assenza QBASIC riserverebbe comunque 11 posizioni di memoria per le componenti del vettore.
Il dimensionamento pone uguale a zero il valore iniziale delle componenti numeriche e imposta a lunghezza nulla le componenti di stringa. Quando si usa un vettore o una matrice, QBASIC riserva uno spazio di memoria anche per la sua componente 0 e quindi, per dimensionare il vettore Nome$ con 40 componenti, è sufficiente scrivere

```
DIM Nome$(39)
```

Di questo fatto è bene tener conto per non sprecare inutilmente spazio di memoria, e far partire tutti gli indici da 0. In alternativa si può inserire prima della DIM l'istruzione

```
OPTION BASE 1
```

che fa iniziare da 1 tutti gli indici di vettori e matrici dimensionati successivamente.

OPZIONE DELL'ISTRUZIONE DIM

L'istruzione DIM è in effetti più complessa di quanto abbiamo visto nel paragrafo precedente. Infatti, mentre l'istruzione OPTION BASE permette di cambiare in 1 il primo indice dei vettori, l'istruzione DIM ha un'opzione che permette di assegnare all'indice di partenza un valore qualsiasi, compresi i numeri negativi.
La forma sintattica completa di DIM è la seguente:

```
DIM vettore(indice inferiore TO indice superiore)
```

dove l'*indice inferiore* e l'*indice superiore* possono variare da -32768 a 32767, e il primo deve essere minore del secondo. Il numero *n* di componenti dimensionate da DIM si calcola con la formula:

```
n = indice superiore - indice inferiore + 1
```

per cui, ad esempio, l'istruzione

```
DIM Suff(6 TO 10)
```

riserva spazio per le 10 - 6 + 1 = 5 componenti Suff(6) Suff(7) Suff(8) Suff(9) Suff(10)

CANCELLAZIONE DI UN VETTORE (ERASE)

Nel corso di un programma non è possibile cambiare il numero di componenti di un vettore già dimensionato; infatti ciò provocherebbe un messaggio di errore. Se è proprio necessario ridimensionare un vettore, si può ricorrere all'artificio di cancellarlo con l'istruzione

```
ERASE vettore [...]
```

dove al solito i puntini indicano che si possono cancellare con una stessa istruzione più vettori e/o matrici, separando i loro nomi con la virgola, e quindi ridimensionarli nuovamente. ERASE è anche utile per azzerare i vettori numerici e impostare a lunghezza nulla quelli di stringa, senza dover scrivere un ciclo che esegua questa operazione sulle singole componenti.

ASSEGNAZIONE DI VALORI

Come ho accennato, l'uso di vettori o matrici al posto di variabili senza indici semplifica diverse fasi della scrittura di un programma, a cominciare dalla richiesta dei valori che l'utente deve immettere da tastiera. Nel caso del vettore Nome$, tale richiesta può essere eseguita da questo semplice ciclo:

```
FOR M% = 0 TO 39
 PRINT "Scrivi il nome numero "; M% + 1
 INPUT Nome$(M%)
NEXT M%
```

che chiede all'utente indici che variano da 1 a 40, mentre usa per il vettore Nome$ componenti di indici che variano da 0 a 39. In modo analogo, se si vuole riempire la matrice Temp riga per riga si possono usare i due seguenti cicli nidificati:

```
FOR Ore% = 0 TO 23
 FOR Gior% = 1 TO 7
  PRINT "Temperatura alle ore"; Ore%; " del giorno"; Gior%
  INPUT Temp(Ore%, Gior%)
 NEXT Gior%
NEXT Ore%
```

Anche la visualizzazione o stampa delle componenti di un vettore è molto semplice, potendo essere eseguita dallo stesso ciclo FOR...NEXT di prima o dal seguente ciclo DO...LOOP:

```
M% = 0
DO
 PRINT "Nome numero "; M%; Nome$(M%)
 M% = M% + 1
LOOP UNTIL M% > 39
```

Lo stesso discorso vale per la visualizzazione o stampa delle componenti di una matrice, per la quale sarà necessario nidificare due cicli l'uno dentro l'altro:

```
Ore% = 0
 DO
  Gior% = 1
   PRINT "Temperatura alle ore"; Ore%; " del giorno";
Gior%
  Gior% = Gior% + 1
 LOOP UNTIL Gior% > 7
 Ore% = Ore% + 1
LOOP UNTIL Ore% > 23
```

DIMENSIONAMENTO DINAMICO

Se il numero massimo di elementi che avrà un vettore può variare da un'esecuzione all'altra dello stesso programma, si può eseguire il dimensionamento *dinamico* del vettore, chiedendone il numero di elementi all'utente ogni volta che inizia l'esecuzione del programma. La tecnica è mostrata nel seguente frammento di programma:

```
INPUT "Quanti nomi si vogliono digitare"; N%
DIM Nome$(N%)
FOR K% = 1 TO N%
 INPUT "Nome "; Nome$(K%)
NEXT K%
```

Questo programma richiede però che l'utente conosca in anticipo il numero di elementi che intende immettere. Dato che in genere ciò non succede, è preferibile seguire un'altra tecnica, che richiede all'utente di premere il tasto <Invio> anziché digitare un dato quando ha terminato di scrivere i dati. Ciò si ottiene con il seguente programma, che permette di digitare un numero massimo di 500 nomi

```
DIM Nome$(500)
K% = 1
DO
 PRINT "Scrivi un nome (o premi <Invio> per finire)":
 INPUT "", Nome$(K%)
 IF Nome$(K%) = "" THEN EXIT DO
 K% = K% + 1
LOOP UNTIL K% > 500
```

INTERPOLAZIONE DI ORDINE N

Una delle applicazioni informatiche più semplici dei vettori è costituita dall'**interpolazione**, che permette di risolvere il seguente problema. Supponiamo di avere una tabella simile alla Tabella 15, che riporta nella colonna di sinistra alcuni numeri interi e in quella di destra le loro radici quadrate.

numero (variabile X)	**radice quadrata (variabile Y)**
1	1
4	2
9	3
16	4
25	5
36	6

***Tabella 15** - Tabella che riporta alcuni numeri e le loro radici quadrate*

Può essere necessario conoscere la radice quadrata di un numero che non si trovi nella prima colonna o, come si dice in termini più matematici, "il valore della variabile Y che corrisponde a un valore della variabile X" (se il numero è *compreso* fra quelli elencati si parla di *interpolazione,* altrimenti di *estrapolazione,* ma i due problemi sono sostanzialmente analoghi).

Tale ricerca risulta tanto più precisa quanto più numerose sono le coppie di numeri di cui si dispone, e in particolare si parla di interpolazione *lineare* se le coppie sono *due,* di interpolazione *quadratica* se le coppie sono *tre*.
La ragione di questa terminologia sta nel fatto che le coppie di valori x, y si possono interpretare come coordinate di punti sul piano, e due punti individuano una linea retta, tre punti individuano una parabola di secondo grado o quadratica, e così via. In ogni caso il valore incognito della variabile Y viene considerato come la coordinata verticale di un punto che si trovi sulla curva individuata dagli altri punti di cui si conoscono le coordinate. La Figura 17 mostra la rappresentazione grafica dell'interpolazione lineare, in cui sono note due coppie di valori o punti: (X_1, Y_1) e (X_2, Y_2), e si cerca la *y* che corrisponde a una determinata *x*

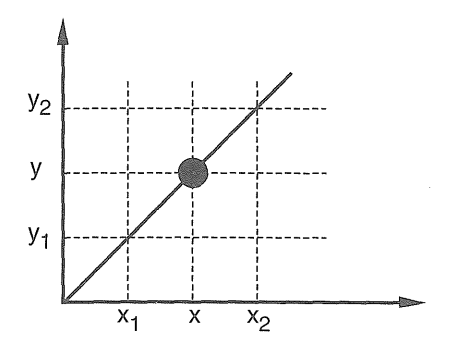


Figura 17 *- Ricerca di un valore y tramite l'interpolazione lineare*

La formula che permette di calcolare la *y* dipende dal numero di punti che si intendono utilizzare: in Figura 18 ho riportato quella valida per 3 punti (interpolazione quadratica), ma questa formula può essere estesa facilmente, dopo una breve riflessione, a un numero di punti qualsiasi.

$$y = \frac{(x - x2)(x - x3)}{(x1 - x2)(x1 - x3)} y1 + \frac{(x - x1)(x - x3)}{(x2 - x1)(x2 - x3)} y2 = \frac{(x - x1)(x - x2)}{(x3 - x1)(x3 - x2)} y3$$

Figura 18 *- Formula dell'interpolazione quadratica*

Il programma INTERPOLA che propongo usa appunto questa formula generalizzata per eseguire l'interpolazione con un numero di punti qualsiasi, le cui coordinate sono memorizzate nei vettori X, Y che vengono dimensionati dinamicamente.
A proposito di questo programma, possiamo osservare che i dati immessi dall'utente sono dapprima memorizzati in variabili di stringa (N$, X$, Y$), che vengono quindi convertite in numeri, secondo la tecnica raccomandata nel paragrafo "Immissione di dati dalla tastiera" (→ pag. 55). Ciò per evitare che si immettano per errore dei dati non numerici, nel qual caso il programma tornerebbe a chiedere di immettere i dati corretti; (naturalmente, con questa tecnica non è possibile immettere come dato lo zero).

```
REM ***INTERPOLA***
CLS
10 INPUT "Numero dei punti"; N$
N% = VAL(N$): IF N% = 0 THEN 10
DIM X(N%), Y(N%)
FOR K% = 1 TO N%
20 PRINT "Scrivi x"; K%; ", y"; K%; ", separandoli
con una virgola"
 INPUT X$, Y$
 X(K%) = VAL(X$): Y(K%) = VAL(Y$)
 IF X(K%) = 0 OR Y(K%) = 0 THEN GOTO 20
NEXT K%
INPUT "Scrivi x"; X
FOR M% = 1 TO N%
 N = 1: D = 1
 FOR L% = 1 TO N%
  IF L% = M% THEN L% = L% + 1: IF L% > N% THEN 30
  N = N * (X - X(L%))
  D = D * (X(M%) - X(L%))
 NEXT L%
30 Y = Y + N * Y(M%) / D
NEXT M%
PRINT "y ="; Y
```

Per farci un'idea del tipo di precisione ottenibile con questo metodo, possiamo immaginare di calcolare la radice quadrata di 9, immettendo prima due, poi tre e infine quattro coppie di valori. I risultati che si ottengono sono riportati in Tabella 16.

se si immettono le coppie di valori...	**si ottiene per la radice quadrata di 9 il valore...**
(2,4) (4,16)	2.833333
(1,1) (2,4) (4,16)	3.222222
(2,4) (4,16) (5,25)	2.925926
(1,1) (2,4) (4,16) (5,25)	3.123457

Tabella 16 *- I valori di approssimazioni successive per calcolare una radice quadrata con il metodo dell'interpolazione*

LETTURA E ASSEGNAZIONE DI VALORI (READ...DATA)

Quando si scrive un programma, se si conoscono in partenza i valori che assumeranno le variabili impiegate, conviene assegnare tali valori da programma stesso, piuttosto che farli immettere da tastiera dall'utente. A questo scopo, oltre all'istruzione LET già vista nel paragrafo "Assegnazione di valori da programma" (→ pag. 19), si può usare la coppia di istruzioni READ...DATA, più convenienti soprattutto nel caso di assegnazione di valori a vettori e matrici.
Le istruzioni READ...DATA operano sempre in coppia, secondo la seguente forma sintattica:

```
READ variabile[...]
DATA costante[...]
```

dove: la *variabile* può essere di uno qualsiasi dei tipi accettati da QBASIC (vedi paragrafo "Tipi di variabili" a pag. 20"); la *costante* deve essere dello stesso tipo della *variabile,* e se è di stringa deve essere racchiusa tra virgolette; i puntini indicano la possibilità di scrivere sulla stessa riga più variabili e costanti, separandole con ",".
L'istruzione READ *legge* le costanti indicate nell'istruzione DATA e le assegna ordinatamente alle proprie variabili; perciò è necessario che ogni *variabile* indicata in READ sia dello stesso tipo della corrispondente *costante* di DATA (altrimenti compare un messaggio di errore di sintassi).
Ad esempio, la coppia di istruzioni:

```
READ K%, Quoz, Risult$
DATA 5, 29.18, "Esatto"
```

ha l'effetto di assegnare alla variabile K% il valore 5, a Quoz il valore 29.18 e a Risult$ il valore "Esatto". Essa è quindi equivalente alle istruzioni

```
K% = 5
Quoz = 29.18
Risult$ = "Esatto"
```

A questo proposito osserviamo che:

- è completamente equivalente scrivere le costanti da assegnare alle variabili in una sola o in più istruzioni DATA; infatti tutte le istruzioni DATA che compaiono in un programma sono considerate come un'unica istruzione. Perciò la precedente coppia di istruzioni READ...DATA potrebbe anche essere sostituita dalle seguenti:

```
READ K%, Quoz, Risult$
DATA 5
DATA 29.18
DATA "Esatto"
```

- le istruzioni READ e DATA si possono trovare in punti qualsiasi del programma, anche distanti fra loro; tuttavia, dato che QBASIC ricerca le istruzioni DATA a partire dall'inizio del programma, è bene che esse si trovino in questo punto, per un'esecuzione più veloce.

RILETTURA DI COSTANTI (RESTORE)

Quando incontra per la prima volta un'istruzione READ, QBASIC cerca nel programma, a partire dall'inizio, una o più istruzioni DATA, i cui valori assegna ordinatamente alle variabili indicate in READ. Se la o le istruzioni DATA contengono più costanti di quante siano le variabili indicate nell'istruzione READ, QBASIC mantiene traccia dell'ultima costante assegnata, in modo che un'eventuale successiva istruzione READ inizi l'assegnazione a partire dalla prima costante non assegnata. Quindi la coppia di istruzioni:

```
READ K%, Quoz, Risult$
DATA 5, 29.18, "Esatto"
```

è completamente equivalente alle seguenti

```
READ K%
READ Quoz
READ Risult$
DATA 5, 29.18, "Esatto"
```

Tuttavia, la lettura delle costanti da parte di un'istruzione READ può cominciare anche dalla prima istruzione DATA del programma, se si premette a READ l'istruzione

```
RESTORE
```

oppure da una *riga* di etichetta prefissata, se si premette a READ l'istruzione

RESTORE *riga*

Ciò è dimostrato dal seguente programma:

```
DATA "Confermi?", 5000, "Esatto"
READ Dom$, Tot%
RESTORE
READ Risult$
PRINT Dom$, Tot%, Risult$
```

che produce l'uscita

```
Confermi?      5000          Confermi?
```

Se le costanti complessivamente indicate nelle istruzioni DATA sono in numero maggiore delle variabili delle istruzioni READ, le costanti in più sono ignorate.

TECNICHE DI LETTURA DEI DATI

Un vantaggio delle istruzioni READ...DATA è che esse permettono di inserire o togliere facilmente dati aventi la stessa struttura. Ad esempio, per leggere e visualizzare i cognomi di un certo numero di studenti, seguiti dalla rispettiva media e numero di assenze, si può inserire la coppia di istruzioni:

```
READ Cognome$, Media%, Assen%
PRINT Cognome$, Media%, Assen%
```

in un ciclo che inizia con l'etichetta

```
Prosstud:
```

termina con l'istruzione

```
GOTO Prosstud
```

ed è seguito da istruzioni del tipo:

```
DATA "Derossi", 7, 20
DATA "Maresca", 6, 15
```

Tuttavia è necessario che il ciclo di lettura termini dopo aver letto tutti i dati; ciò si ottiene scrivendo dopo l'ultima riga di dati una riga del tipo

```
DATA "ZZ", 99, 99
```

(o contenente un qualsiasi altro valore che non sarà sicuramente posseduto dal Cognome), e inserendo nel ciclo di lettura un'istruzione di controllo del tipo

```
IF Cognome$ = "ZZ" THEN GOTO Basta
```

che rimanda a linee del tipo:

```
Basta:
 END
```

Il programma completo è pertanto il seguente LEGGERE AD.
In esso, come ho accennato, si possono aggiungere quante linee dati si vogliono, purché abbiano la stessa struttura di quelle esistenti, senza dover apportare alcuna modifica al resto del programma.

```
REM ***LEGGEREAD***
CLS
Prosstud:
 READ Cognome$, Media%, Assen%
 IF Cognome$ = "ZZ" THEN GOTO Basta
 PRINT Cognome$, Media%, Assen%
 GOTO Prosstud
DATA "Derossi", 7, 20
DATA "Maresca", 6, 15
DATA "ZZ", 99, 99
Basta:
 END
```

RICERCA DELL'ELEMENTO MASSIMO DI UN VETTORE

In questo e nel paragrafo successivo illustrerò due esempi di elaborazioni molto frequenti su insiemi di dati numerici o alfabetici, che si trovino contemporaneamente presenti nella memoria di un computer sotto forma di componenti di un vettore.

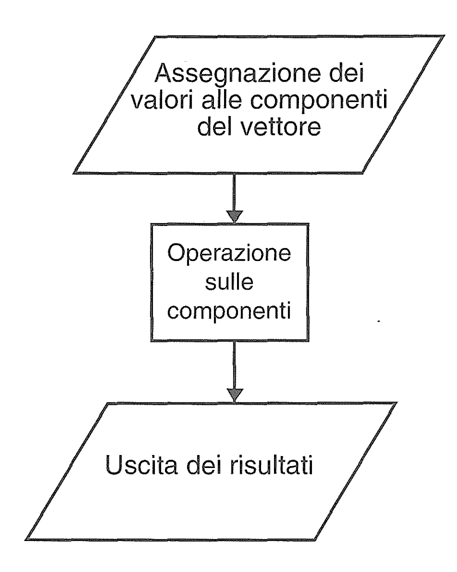


Figura 19 *- Le tre fasi dell'elaborazione di un vettore*

In entrambi i casi l'elaborazione consisterà nelle tre fasi schematizzate in Figura 19, dove in particolare l'assegnazione dei valori alle componenti viene effettuata dal programma stesso tramite istruzioni READ...DATA. Ciò al solo scopo di poter presentare al lettore anche il tipo di risultati che il programma produce sullo schermo.

Se invece i dati vengono immessi tramite tastiera dall'utente, sarà sufficiente modificare l'assegnazione dei valori con uno dei cicli contenenti l'istruzione INPUT che abbiamo visto in precedenza.
Per trovare l'elemento massimo di un insieme numerico è sufficiente confrontare ciascun elemento con il primo e, se si trova un elemento maggiore, questo diventa la base per i confronti successivi. Si continua cosi finché si raggiunge la fine del vettore.
Il seguente programma TROVAMASS ricerca e stampa il valore massimo fra quelli delle tre vendite memorizzate nel vettore Vend.

```
REM ***TROVAMASS***
DATA 1500, 7200, 4800, -99
DIM Vend(2)
 K% = 1
 DO
  READ Vend(K%)
  K% = K% + 1
 LOOP UNTIL Vend(K% - 1) = -99
 K% = K% - 2
 Vendmax = Vend(1)
 FOR M% = 2 TO K%
  IF Vend(M%) > Vendmax THEN
   Vendmax = Vend(M%)
  END IF
 NEXT M%
 CLS
 PRINT "La vendita massima è: "; Vendmax
```

Esso produce l'uscita:

```
La vendita massima è: 7200
```

ORDINAMENTO DELLE COMPONENTI DI UN VETTORE (SWAP)

Un'altra operazione molto frequente su dati numerici o alfabetici è quella di ordinarli in verso crescente o in ordine alfabetico. I due problemi sono completamente equivalenti per un computer che, come già sappiamo, tratta come numeri sia i dati alfabetici sia quelli numerici (tramite la conversione in codici ASCII vista al paragrafo "Tabella dei codici ASCII"), → pag. 14. I programmi che risolvono il problema dell'ordinamento o *sort* delle componenti di un vettore si basano su diversi approcci, i più noti dei quali sono il *bubble sort,* il *quick sort e* lo *shell sort.* Tutti questi metodi confrontano ciascuna componente del vettore con un'altra, e ne scambiano i valori se non sono già nell'ordine voluto. Quest'operazione è eseguita dall'istruzione:

```
SWAP var1, var2
```

inserita in un ciclo IF...END IF, che scambia fra loro i valori delle variabili *var1* e *var2*. Il semplice programma che segue dispone in ordine crescente una lista di numeri basandosi sulla tecnica del *bubble sort;* essa è così chiamata perché alla fine di ogni ciclo il valore più piccolo si porta "in cima" agli altri, come fanno le bolle d'aria in un bicchiere d'acqua.

```
REM ***ORDINA***
DATA 5, 40, 71, 8, 27, 32, 49, 60, 56, 19
DIM Num(9)
CLS
PRINT "Numeri non ordinati:"
FOR K% = 0 TO 9
 READ Num(K%)
 PRINT Num(K%);
NEXT K%
FOR L% = 0 TO 9
 FOR M% = L% + 1 TO 9
  IF Num(L%) > Num(M%) THEN
   SWAP Num(L%), Num(M%)
  END IF
 NEXT M%
NEXT L%
PRINT
LINE INPUT "Premi <Invio> per vedere i numeri
ordinati"; R$
FOR N% = 0 TO 9
 PRINT Num(N%);
NEXT N%
```

Esso produce la seguente uscita:

```
Numeri non ordinati:
5  40  71  8  27  32  49  60  56  19

Premi <Invio> per vedere i numeri ordinati
5  8  19  27  32  40  49  56  60  71
```

Come ho accennato, il precedente programma ORDINA funziona anche per ordinare alfabeticamente una lista di parole, pur di sostituire il vettore Numero con un vettore Nome$, (e naturalmente modificare i messaggi informativi sullo schermo).
Permette anche l'ordinamento inverso, sostituendo l'operatore ">" con "<".

ALTRE OPERAZIONI CON I VETTORI

Quando si opera con i vettori, vi sono alcune operazioni abbastanza frequenti che si possono eseguire con i semplici programmi riportati in questo paragrafo.

Somma delle componenti di un vettore
Dato un vettore V con n elementi, memorizzare la loro somma nella variabile S. Il programma è il seguente:

```
S = 0
FOR i% = 1 TO n%
 S = S + V(i%)
NEXT i%
PRINT "La somma è"; S
```

Somma delle componenti di una matrice
Data una matrice M con r righe e c colonne, memorizzare la loro somma nella variabile S. Il programma è il seguente:

```
S = 0
FOR i% = 1 TO r%
  FOR Jù = 1 TO c%
   S = S + M(i%, j%)
  NEXT j%
next i%
PRINT "La somma è"; S
```

Somma delle componenti di una matrice per righe
Data una matrice M con r righe e c colonne, sommare gli elementi delle sue righe, memorizzando i risultati nelle componenti del vettore S. Il programma è il seguente (dove ho omesso le istruzioni di dimensionamento):

```
FOR i% = 1 TO r%
 FOR j% = 1 TO c%
  S (i%) = S(i%) + M(i%, j%)
 NEXT j%
NEXT i%
FOR i% = 1 TO r%
 PRINT "La somma della riga"; i%; "è"; S(i%)
NEXT i%
```

Capitolo 11

Programmazione modulare



CHE COSA SONO I MODULI DI PROGRAMMA

I programmi visti finora erano piuttosto corti, dato che ciascuno si proponeva di illustrare una singola tecnica di programmazione. Tuttavia, i programmi effettivi consistono normalmente in molte pagine di codice, e quindi risulta difficile organizzarli e comprenderli, specie dopo un certo tempo che sono stati scritti.
Per ovviare a questa difficoltà è stata sviluppata la tecnica della **programmazione modulare;** essa consiste nello spezzare un lungo programma in più moduli o blocchi di istruzioni, ciascuno dei quali è in grado di eseguire un singolo aspetto del problema che si vuole risolvere.
Per spiegarmi con un paragone, dirò che impiega di solito una tecnica modulare chi si appresta a scrivere un libro. Infatti, il compito complesso

SCRIVI IL LIBRO

si può spezzare in un certo numero di compiti più semplici, quali

SCRIVI L'INTRODUZIONE
SCRIVI IL 1° CAPITOLO
SCRIVI IL 2° CAPITOLO
...
SCRIVI L'INDICE

Alcuni di questi compiti si possono a loro volta spezzare in passi più semplici, quali

SCRIVI IL 1° PARAGRAFO
SCRIVI IL 2° PARAGRAFO

e così via.
Se rappresentiamo graficamente questo tipo di organizzazione, come in Figura 20, lo schema che otteniamo assomiglia a un triangolo (o a una piramide), con il compito principale (SCRIVI IL LIBRO) situato nel vertice, e quelli di livello via via più semplice al di sotto. Per questa ragione la programmazione modulare è detta anche **approccio top-down,** che significa "dall'alto verso il basso".
QBASIC mette a disposizione tre tipi di "mattoni" o moduli per costruire questa piramide: le subroutine, i sottoprogrammi e le funzioni.

SCRIVI IL LIBRO

SCRIVI LA INTRODUZIONE | SCRIVI IL 1° CAPITOLO | SCRIVI IL 2° CAPITOLO

SCRIVI IL 1° PARAGRAFO | SCRIVI IL 2° PARAGRAFO | SCRIVI IL 3° PARAGRAFO | SCRIVI IL 1° PARAGRAFO | SCRIVI IL 2° PARAGRAFO

***Figura 20** - Suddivisione di un compito complesso in moduli più semplici*

SUBROUTINE (ON...GOSUB RETURN)

Una prima possibilità piuttosto rudimentale di programmazione modulare, già offerta dalle precedenti versioni di BASIC e mantenuta per ragioni di compatibilitàda QBASIC, è la struttura di decisione a scelta multipla ON...GOSUB, che ha un effetto simile all'istruzione SELECT CASE già vista.
In particolare, ON...GOSUB, che ha la forma sintattica

```
ON espressione GOSUB lista
```

dove: il valore dell'*espressione* numerica deve essere compreso fra 0 e 255, e la *lista* è un elenco di numeri di riga di espressioni eseguibili. ON...GOSUB esamina il valore dell'*espressione* e, se lo trova

- uguale a 1, esegue il blocco di istruzioni che inizia al primo numero di riga indicato nella *lista;*
- uguale a 2, esegue il blocco di istruzioni che inizia al secondo numero di riga indicato e così via.

Ciascun blocco di istruzioni termina con la parola chiave

```
RETURN
```

che determina la prosecuzione dell'elaborazione a partire dall'istruzione che segue l'istruzione ON...GOSUB.
Secondo una terminologia molto diffusa, i blocchi di istruzioni compresi fra il numero di riga e la parola chiave RETURN sono detti **subroutine**, mentre le istruzioni di programma diverse da quelle che costituiscono le subroutine sono dette **programma principale.**
Le subroutine si possono trovare in punti qualsiasi del programma principale, anche se è preferibile che siano raggruppate tutte alla fine; in tal caso il programma principale deve contenere istruzioni che lo separino dalle subroutine, per evitare che le loro istruzioni possano essere eseguite al di fuori dei richiami espliciti contenuti nell'istruzione

ON...GOSUB

Allo scopo si potrà usare l'istruzione

END

prima della prima subroutine, oppure racchiudere il programma principale in una opportuna struttura iterativa, come mostra il seguente programma SCELTA1.
In questo caso in risposta a un numero immesso dall'utente viene eseguita un'opportuna subroutine: i numeri 1, 2, 3 fanno eseguire quella che inizia a riga 50, i numeri 4, 5 quella che inizia a riga 100.

```
REM ***SCELTA1***
X% = -1
WHILE X%
 INPUT "Fai la tua scelta (0 per finire)", X%
 IF X% = 0 THEN END
 WHILE X% < 1 OR X% > 5
  PRINT "Il valore dev'essere compreso fra 1 e 5"
  INPUT "Fai la tua scelta (0 per finire)", X%
 WEND
 ON X% GOSUB 50, 50, 50, 100, 100
WEND
50 PRINT "Subroutine 50"
RETURN
100 PRINT "Subroutine 100"
RETURN
```

CONFRONTO FRA ON...GOSUB E SELECT CASE

Come ho osservato, ON...GOSUB è un residuo delle precedenti versioni del BASIC, e viene di solito sostituita con la nuova istruzione SELECT CASE, che risulta più versatile e presenta i seguenti vantaggi:

- L'*espressione* contenuta in SELECT CASE è una variabile di tipo o numerico o di stringa, mentre l'*espressione* contenuta in ON...GOSUB deve essere una variabile numerica con valore tra 0 e 255.
- L'istruzione SELECT CASE fa diramare a un blocco di programma situato subito dopo la clausola CASE che è stata verificata, mentre ON...GOSUB fa diramare a una subroutine situata in un altro punto del programma.
- La clausola CASE permette di controllare una *espressione* con un intervallo di valori, e se questo è piuttosto ampio il confronto eseguito con ON...GOSUB risulta alquanto laborioso. Perciò il programma scelta 1 si può riscrivere, in maniera più chiara, usando l'istruzione SELECT CASE, come indicato nel successivo programma SCELTA 2.

```
REM ***SCELTA2***
DO
 INPUT "Fai la tua scelta (0 per finire)", X%
 SELECT CASE X%
  CASE 0
   END
  CASE 1 TO 3
   PRINT "Subroutine 50"
  CASE 4 TO 5
   PRINT "Subroutine 100"
  CASE ELSE
   PRINT "Il valore dev'essere compreso fra 1 e 5"
  END SELECT
LOOP
```

PROCEDURE

Dopo le subroutine, gli altri due tipi di moduli di programma sono costituiti dai sottoprogrammi e dalle funzioni che, poiché si comportano per molti aspetti nello stesso modo, vengono detti globalmente **procedure.**
In particolare i sottoprogrammi e le funzioni sono definiti secondo la medesima sintassi, che ha la forma generale indicata in Tabella 17.

Sottoprogrammi	Funzioni
SUB *nome [parametri]* [STATIC] . . [EXIT SUB] . . END SUB	FUNCTION *nome [parametri]* [STATIC] . . [EXIT FUNCTION] . . END FUNCTION

Tabella 17 *- Confronto tra la sintassi dei sottoprogrammi e delle funzioni*

In essa: il *nome* può essere lungo fino a 40 caratteri, i *parametri* sono una lista di variabili (separate da virgole) impiegate all'interno della procedura o *locali,* l'attributo STATIC fa sì che le variabili locali mantengano i loro valori nel caso di successive chiamate della procedura.
In assenza dell'attributo STATIC le variabili sono considerate *automatiche,* cioè vengono inizializzate a zero o a stringhe nulle ogni volta che la procedura è chiamata. È tuttavia possibile usare l'attributo STATIC all'interno della procedura per rendere alcune variabili automatiche e altre statiche, come vedremo nel successivo paragrafo "Variabili statiche e automatiche" (→ pag. 120).
Le istruzioni opzionali EXIT SUB ed EXIT FUNCTION permettono di uscire anticipatamente da una procedura, di solito al verificarsi di una certa condizione.
All'interno di una procedura non si possono usare le seguenti istruzioni:

COMMON, DECLARE, DIM SHARED, OPTION BASE,
TYPE...END TYPE, DEF FN, FUNCTION, SUB.

Perciò non è possibile l'annidamento di procedure, né la definizione di funzioni dentro una procedura; tuttavia una procedura può richiamarne un'altra o una funzione DEF FN, come vedremo nel paragrafo "Procedure ricorsive" (→ pag. 123).

SOTTOPROGRAMMI (CALL, SUB...END SUB)

I **sottoprogrammi** sono una specie di piccoli programmi all'interno del programma principale, e vengono da questo chiamati con l'istruzione CALL. Esempio:

```
CALL Freccia (X%)
```

Per vedere un semplice esempio di sottoprogramma, supponiamo di voler scrivere un programma che faccia muovere una piccola freccia (>—->) dalla sinistra alla destra dello schermo.
Il programma potrebbe avere la struttura del seguente listato PROVASUB

```
REM *** PROVASUB ***
CLS
FOR X% = 1 TO 70
 CALL Freccia (X%)
NEXT X%

SUB Freccia (X%)
 LOCATE 10, X%
 PRINT ">—->"
 FOR pausa = 1 TO 100: NEXT pausa
 LOCATE 10, X%
 IF X% < 70 THEN PRINT "      "
END SUB
```

Le prime quattro righe costituiscono il programma principale, che dopo aver pulito lo schermo richiama 70 volte il sottoprogramma Freccia, con un ciclo FOR...NEXT. In altri termini (vedi Figura 21), l'istruzione

```
CALL Freccia (X%)
```

fa eseguire le istruzioni del sottoprogramma comprese fra `LOCATE 5, X%` e `IF X% < 70 THEN PRINT " "`, mentre l'istruzione
`END SUB`
rimanda al programma principale, facendo eseguire l'istruzione `NEXT X%`.

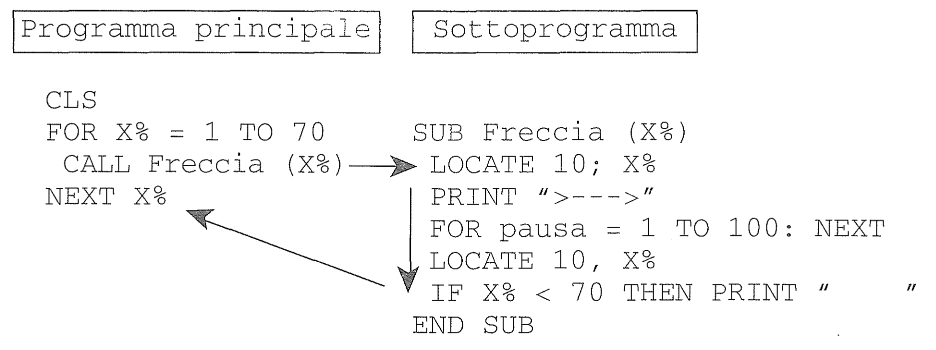


***Figura 21** - Ordine di esecuzione delle istruzioni in presenza di un sottoprogramma*

La X% che compare nella definizione della procedura Freccia (linea `SUB...`) è detta suo *parametro,* mentre la X% che compare nella chiamata della procedura (linea `CALL...`) è detta suo *argomento.*
Si dice che il programma principale fornisce o *passa* l'argomento al parametro, che in questo caso determina la colonna di schermo nella quale viene disegnata la freccia (la riga è sempre la 10). Il valore del parametro X% cambia costantemente da 1 a 70 per produrre l'effetto di movimento; affinché questo sia possibile è però ancora necessario:

- cancellare la freccia appena disegnata prima di disegnare quella successiva. Ciò è eseguito dalla coppia di istruzioni

```
LOCATE 10, X%
IF X% < 70 THEN PRINT " "
```

- lasciare sullo schermo per un certo tempo la freccia appena disegnata, prima di cancellarla. Ciò si può ottenere con il ciclo di ritardo

```
FOR pausa = 1 TO 100: NEXT pausa
```

  (ma nel paragrafo "Pausa in un programma con DO...LOOP", → pag. 122 vedremo un'alternativa migliore).

Una procedura può avere più parametri, che vanno separati con virgole; bisogna però fare attenzione al fatto che gli argomenti indicati nella linea del programma principale che richiama la procedura (linea CALL...) siano dello stesso tipo e nello stesso ordine dei parametri indicati nella prima riga della procedura (linea SUB...). Se quindi si modifica la precedente chiamata in

```
CALL Freccia (x%, y!)
```

anche la prima linea della procedura va modificata in

```
SUB Freccia (x%, y!)
```

Tuttavia, non è necessario che i nomi di variabile usati nella linea SUB siano gli stessi della linea CALL, ma solo che indichino variabili dello stesso tipo. Quindi, nell'esempio precedente, la linea SUB potrebbe anche essere scritta

```
SUB Freccia (riga%, colonna!)
```

e naturalmente la procedura dovrebbe usare al suo interno queste ultime variabili (riga%, colonna!) anziché quelle della chiamata (x%, y!).
Sui sottoprogrammi vedremo altre informazioni dopo aver parlato delle funzioni.

EDITAZIONE DI UN SOTTOPROGRAMMA

Osserviamo che in fase di editazione, quando si preme il tasto <Invio> dopo avere scritto l'istruzione *SUB Freccia (X%),* lo schermo di QBASIC visualizza le seguenti linee:

```
SUB Freccia (x%)

END SUB
```

Viene cioè inserita automaticamente la linea `END SUB`, mentre nel titolo della finestra superiore compare il nome del sottoprogramma (Freccia). Dopo aver scritto le altre istruzioni del sottoprogramma, si può salvare l'intero programma con il nome PROVASUB (l'estensione .BAS viene inserita automaticamente da QBASIC) ed eseguirlo premendo il tasto <F5>.
Per rivedere sullo schermo il listato del programma principale e i moduli che lo costituiscono, si può:

- selezionare la voce SUB nel menu Visualizza, oppure
- premere il tasto <F2>

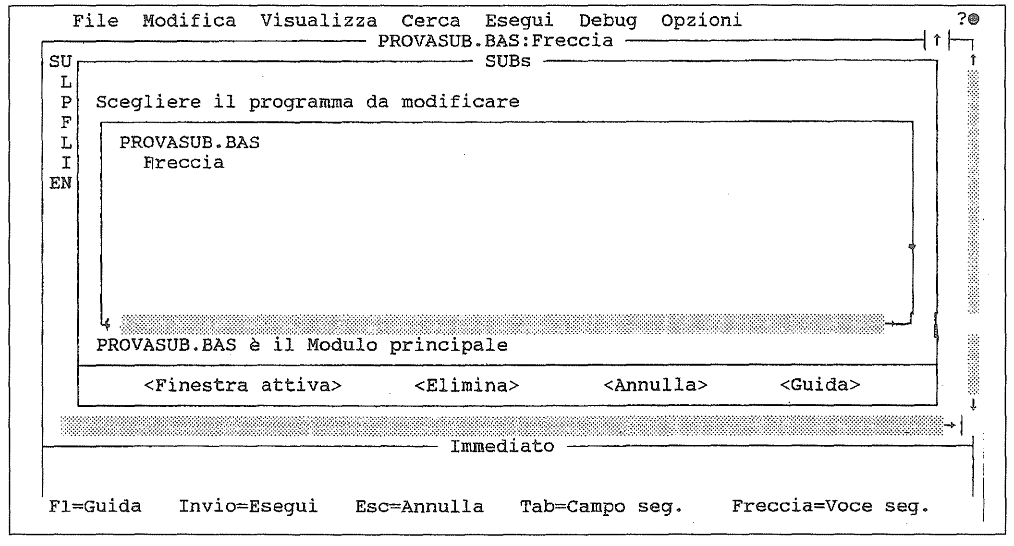


***Figura 22** - Videata con i titoli del programma principale e dei suoi sottoprogrammi*

Compare la videata di Figura 22, che permette di richiamare un modulo qualsiasi, ed eventualmente annullarlo.
Se si richiama il modulo principale, si vede che ora esso inizia con la linea

```
DECLARE SUB Freccia (x%)
```

aggiunta anch'essa da QBASIC, quando il programma è stato salvato.

FUNZIONI (FUNCTION...END FUNCTION)

Le funzioni sono anch'esse una specie di piccoli programmi all'interno del programma principale ma, diversamente dai sottoprogrammi, restituiscono sempre un valore, numerico o di stringa, al programma principale, assegnandolo a una variabile che ha lo stesso nome della funzione.
Altra differenza con i sottoprogrammi è che una funzione definita dall'utente viene chiamata dal programma principale nella stessa maniera delle funzioni incorporate di QBASIC (quali quelle numeriche e di stringa, viste nei capitoli 6 e 7), cioè usando il suo nome in un'espressione. Ad esempio, la funzione Richiesta$ potrebbe venire chiamata dal programma principale in uno dei seguenti modi:

```
PRINT Richiesta$
Ri$ = Richiesta$
```

Scriviamo come esempio un programma che chieda all'utente di digitare il suo nome, e quindi calcoli tramite una funzione il numero dei caratteri che ha digitato. Il listato potrebbe essere quello di Figura 23.

```
CLS
Ri$ = Richiesta$
LOCATE 2, 1
PRINT "Il tuo nome è lungo"; LEN(Ri$); "caratteri"

FUNCTION Richiesta$
 LOCATE 20, 1
 PRINT "Scrivi qui sotto il tuo nome"
 PRINT "---------------"
 PRINT ">"
 PRINT "---------------"
 LOCATE 22, 3
 LINE INPUT r$
 Richiesta$ = r$
END FUNCTION
```

***Figura 23** - Listato di un programma che usa una funzione definita dall'utente*

Il programma principale, dopo avere pulito lo schermo, chiama la funzione Richiesta$, che chiede all'utente di digitare il suo nome e lo memorizza nella variabile r$. L'istruzione di assegnazione

```
Richiesta$ = r$
```

fa sì che la stringa scritta dall'utente sia restituita dalla funzione al programma principale. Una volta ricevuta la stringa, il programma principale ne indica la lunghezza, con un messaggio posizionato tramite l'istruzione LOCATE.

Anche in questo caso QBASIC inserisce automaticamente la riga

```
END FUNCTION
```

in fase di editazione e, dopo che il programma è stato salvato, la riga

```
DECLARE FUNCTION Richiesta$ ()
```

all'inizio del programma principale.

Nota. Osserviamo che se un modulo di programma chiama una funzione definita in un altro modulo, QBASIC non inserisce automaticamente l'istruzione DECLARE nel modulo chiamante, ma lascia questo compito all'utente. Se un modulo che chiama una funzione definita in un altro modulo non contiene all'inizio l'istruzione DECLARE, la funzione chiamata viene considerata come il nome di una variabile.

CONFRONTO TRA FUNCTION E DEF FN

QBASIC conserva dalle versioni precedenti la possibilità di definire una funzione con l'istruzione DEF FN (potenziata rispetto al passato), oltre che con la nuova istruzione FUNCTION.
Tuttavia, FUNCTION presenta rispetto a DEF FN diversi vantaggi, elencati in Tabella 18 (alcuni di essi saranno spiegati nei paragrafi successivi).

FUNCTION	**DEF FN**
Tutte le variabili sono locali, sebbene sia possibile usare variabili globali	Tutte le variabili sono globali nel modulo corrente, sebbene possano essere rese locali
Il passaggio delle variabili da parte del programma principale avviene per riferimento o per valore (→ pag. 118)	Il passaggio delle variabili da parte del programma principale avviene solo per valore (→ pag. 119)
Permette di scrivere funzioni ricorsive (→ pag. 123)	Non permette di scrivere funzioni ricorsive (pag. 123)
La procedura può essere definita in un modulo e usata nello stesso o in un altro (nel secondo caso va dichiarata in un'istruzione DECLARE, come indicato nel paragrafo precedente)	La funzione deve essere definita e usata nello stesso modulo, e la definizione deve precedere l'uso
Il nome può essere qualsiasi, purché non inizi con le lettere "FN"	Il nome deve iniziare con le lettere "FN"

***Tabella 18** - Differenze tra le istruzioni FUNCTION e DEF FN*

ARGOMENTI E PARAMETRI DELLE PROCEDURE

La precedente funzione Richiesta$ non riceveva dal programma principale valori o argomenti su cui operare, mentre in alcuni casi ciò è necessario. Ad esempio, se si volesse definire una funzione (Potint&) che calcola la potenza intera di un numero intero, impiegando per entrambi dati di tipo intero LONG, si potrebbe scrivere:

```
FUNCTION Potint& (X&, Y&) STATIC
 Pot& = 1
 FOR I& = 1 TO Y&
  Pot& = Pot& * X&
 NEXT I&
 Potint& = Pot&
END FUNCTION
```

Anche in questo caso i nomi di variabili X&, Y& che compaiono nella definizione della funzione dopo il suo nome (Potint&) sono detti *parametri,* e ricevono un valore dal programma principale quando esso chiama la funzione (ad esempio usando il suo nome in un'istruzione PRINT). Quindi, riassumendo:

> Gli **argomenti** sono costanti, variabili o espressioni che vengono passati a una SUB o a una FUNCTION quando la SUB o la FUNCTION vengono chiamate.
> I **parametri** sono dei "segnaposto" per gli argomenti che compaiono in una definizione di procedura.

Come indica la Figura 24, quando si chiama una procedura, gli argomenti vengono inseriti nelle variabili della lista dei parametri, in modo che il primo parametro riceve il primo argomento, il secondo parametro riceve il secondo argomento, e così via.

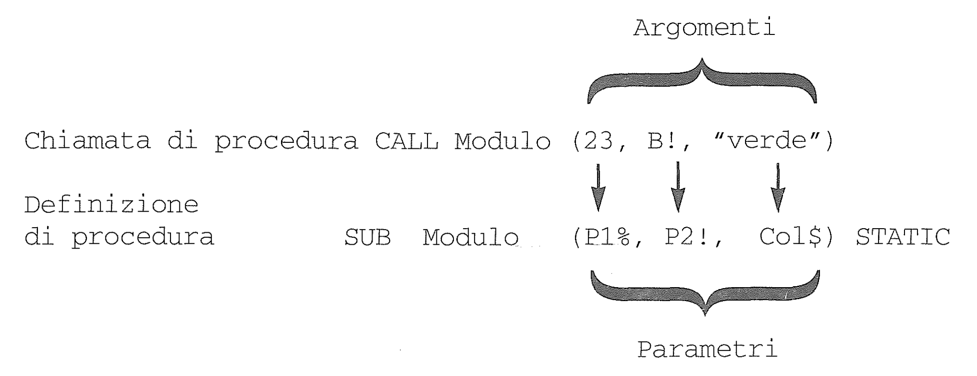


Figura 24 *- Argomenti e parametri di una procedura*

La Figura 24 mostra anche un'altra regola già enunciata: sebbene non sia necessario che i nomi delle variabili siano gli stessi in una lista di argomenti e in una lista di parametri, il numero dei parametri deve essere uguale al numero di argomenti, e anche i tipi degli argomenti e dei parametri che si corrispondono devono essere gli stessi.

Una *lista di argomenti* è costituita da uno o più dei seguenti elementi, separati da ",":

- costanti
- espressioni
- nomi di variabili
- nomi di matrici seguiti da parentesi aperta e chiusa.

Una *lista di parametri* è costituita da uno o più dei seguenti elementi, separati da ",":

- nomi di variabili (quali X$ o X AS STRING), ma non di stringhe a lunghezza fissa (quale X AS STRING * 10)
- nomi di matrici seguiti da parentesi aperta e chiusa.

Ad esempio, la lista dei parametri che compare nella prima linea della definizione di un sottoprogramma potrebbe essere la seguente:

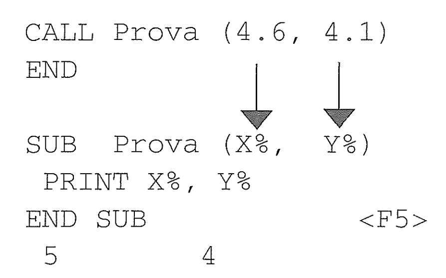
```
SUB Modulo (A%, Matrice(), Varrec AS Tiprec, Col$)
```

In questo caso, il sottoprogramma Modulo potrebbe essere chiamato dall'istruzione CALL del seguente esempio, che gli passa quattro argomenti del tipo corretto:

```
TYPE Tiporec
 Nome AS STRING * 12
 Codice AS LONG
END TYPE
DIM Varrec AS Tiporec
CALL Modulo (X%, M(), Varrec, "codice")
```

PASSAGGIO DI COSTANTI ED ESPRESSIONI

La lista degli argomenti passati a una procedura può contenere delle costanti: naturalmente una costante di stringa deve essere passata a un parametro di stringa, e una costante numerica a un parametro numerico.
Se una costante numerica di una lista di argomenti non è dello stesso tipo del parametro corrispondente nell'istruzione SUB o FUNCTION, la costante viene "forzata" nel tipo del parametro. Esempio:



A una procedura si possono passare anche espressioni risultanti da operazioni su variabili e costanti.
Come nel caso delle costanti, le espressioni numeriche che non si accordano con il tipo dei loro parametri sono forzate in essi.

PASSAGGIO DI ARGOMENTI PER RIFERIMENTO

Come ho accennato al paragrafo "Che cosa sono le variabili e le costanti" (→ pag. 19), ogni variabile di programma ha un proprio indirizzo o locazione di memoria in cui è memorizzata. Quando un programma chiama una procedura e le passa delle variabili, le passa in realtà gli indirizzi che tali variabili hanno in memoria.
Quest'operazione, detta **passaggio per riferimento**, fa sì che tutti i cambiamenti operati da una procedura sui suoi parametri si ripercuotano anche sugli argomenti che il programma principale le ha passato.
Ad esempio il seguente programma PASSRIF, che usa una procedura per cambiare le "e" minuscole di una frase in maiuscole, modifica sia il valore del proprio argomento Prova$ sia del parametro A$ della procedura.

```
REM *** PASSRIF ***
DECLARE SUB Cambia (A$, B$)
Prova$ = "frase con tutte le e minuscole"
PRINT "Prima della chiamata al sottoprogramma:"; Prova$
CALL Cambia(Prova$, "e")
PRINT "Dopo la chiamata al sottoprogramma:"; Prova$
END

SUB Cambia (A$, B$) STATIC
Inizio = 1
DO
 Trova = INSTR(Inizio, A$, B$)
 IF Trova > 0 THEN
  MID$(A$, Trova) = UCASE$(B$)
  Inizio = Inizio + 1
 END IF
LOOP WHILE Trova > 0
END SUB
```

PASSAGGIO DI ARGOMENTI PER VALORE

Se non si vuole che una procedura cambi il valore dell'argomento che le viene passato dal programma principale, è necessario che questo passi il valore contenuto nella variabile, anziché il suo indirizzo.
Quest'operazione, detta **passaggio per valore**, viene eseguita da QBASIC copiando la variabile in una locazione di memoria temporanea, e quindi passando l'indirizzo di questa locazione temporanea; dato che la procedura non conosce l'indirizzo della variabile originaria, non può modificarla e cambia invece la sua copia.
Per passare una variabile per valore è sufficiente racchiuderla tra parentesi nella linea CALL che chiama la procedura, come avviene nel seguente programma passual, che passa A per valore, B per riferimento a un sottoprogramma.

```
REM *** PASSUAL ***
DECLARE SUB Prod (X!, Y!)
A = 1
B = 1
PRINT "Prima della chiamata al sottoprogramma, A ="; A; ", B ="; B
CALL Prod((A), B)
PRINT "Dopo la chiamata al sottoprogramma, A ="; A; ", B ="; B
END

SUB Prod (X, Y) STATIC
 X = 2 * X
 Y = 3 * Y
 PRINT "Nel sottoprogramma, X ="; X; ", Y ="; Y
END SUB
```

Esso produce la seguente uscita, che dimostra quanto detto:

```
Prima della chiamata al sottoprogramma, A = 1 , B = 1
Nel sottoprogramma, X = 2 , Y = 3
Dopo la chiamata al sottoprogramma, A = 1 , B = 3
```

VARIABILI STATICHE E AUTOMATICHE (STATIC)

Come abbiamo visto al paragrafo "Procedure" (→ pag. 93), se si inserisce l'attributo STATIC nella linea di definizione di una procedura, tutte le variabili locali della procedura sono considerate *statiche*, mentre potrebbe essere necessario che alcune di esse rimangano *automatiche*. Per rendere ciò le variabili che si vogliono rendere statiche vengono elencate in un'istruzione STATIC situate nel corpo della procedura, anzichè nella sua definizione. Tutte le altre variabili saranno considerate automatiche. Nel seguente sottoprogramma Prova, entrambe le variabili A$ e B$ sono locali, ma A$ è automatica, cosicché viene reinizializzata a stringa nulla ogni volta che si chiama Prova, mentre B$ è statica, e così conserva il suo valore nelle successive chiamate alla procedura.

```
DECLARE SUB Prova ()
FOR I% = 1 TO 5
 Prova
NEXT I%
END


SUB Prova
 STATIC B$
 A$ = A$ + "*"
 B$ = B$ + "*"
 PRINT A$, B$
END SUB
```

L'uscita del programma è pertanto la seguente:

```
*     *
*     **
*     ***
*     ****
*     *****
```

ÀMBITO DI UNA VARIABILE (DIM SHARED)

Abbiamo visto che il programma principale può fornire o passare a una procedura uno o più argomenti, necessari per il suo funzionamento. Si osservi che il programma PROVASUB visto nel paragrafo "Sottoprogrammi" (→ pag. 111) non funzionerebbe se le istruzioni `CALL Freccia (X%)` e `SUB Freccia (X%)` fossero sostituite rispettivamente da `CALL Freccia` e da `SUB Freccia,` e ciò malgrado il sottoprogramma usi esattamente la stessa variabile X% del programma principale. Ciò perché ogni modulo di programma assegna alle proprie variabili dei valori in modo indipendente da quelli che le stesse variabili possono assumere in altri moduli.

Questo fatto si esprime anche dicendo che una variabile è **locale** al modulo in cui viene definita, o che l'àmbito di una variabile è costituito dal modulo in cui è definita, o che una variabile è visibile solo dal modulo in cui è definita.
Pertanto, le istruzioni

```
CALL Freccia
```
e
```
SUB Freccia
```

determinerebbero la situazione seguente: mentre il programma principale assegna a X% i successivi valori 1, 2, ... 70, nel sottoprogramma X% avrebbe costantemente il valore 0 (che è il valore di inizializzazione di tutte le variabili numeriche, → pag. 19; quindi il tentativo di eseguire l'istruzione `LOCATE 10, 0` determinerebbe un messaggio di errore, in quanto non esiste una posizione di schermo con queste coordinate (→ pag. 25). Se invece si vuole che una variabile usata da un sottoprogramma abbia in esso lo stesso valore che le viene assegnato dal programma principale, occorre che questo passi tale valore al sottoprogramma come suo argomento, come ho già detto varie volte.
Se poi si vuole che una variabile definita nel programma principale sia visibile da *qualsiasi modulo* (o, in altri termini, che il suo àmbito sia costituito dall'intero programma), essa deve essere dichiarata come **globale** (o **condivisa**) nel modulo principale.
Ciò si ottiene con un'istruzione del tipo

```
DIM SHARED X%
```

Il programma PROVASUB di pag. 11 si può quindi riscrivere come indicato in Figura 25.

```
DIM SHARED X%
CLS
FOR X% = 1 TO 70
 CALL Freccia
NEXT X%

SUB Freccia
 LOCATE 10, X%
 PRINT ">-->"
 FOR pausa = 1 TO 100: NEXT pausa
 LOCATE 10, X%
 IF X% < 70 THEN PRINT "     "
END SUB
```

***Figura 25** - Listato di programma che fa muovere una freccia sullo schermo impiegando una variabile globale*

L'uso di variabili globali, se in apparenza semplifica l'attività di programmazione, produce in realtà dei programmi difficili da leggere e da correggere: ad esempio, se una variabile assume un valore che dà luogo a problemi, è necessario esaminare l'intero programma per trovare la causa dell'errore. Pertanto è bene ridurre al minimo il numero di variabili globali, e usare quando possibile la tecnica del passaggio di argomenti alle subroutine.

PAUSA IN UN PROGRAMMA CON DO...LOOP

Come abbiamo visto al paragrafo “Pausa in un programma con FOR...NEXT” (→ pag. 67), l’uso di un ciclo vuoto FOR...NEXT per introdurre una pausa nell’esecuzione di un’istruzione comporta lo svantaggio che la durata della pausa varia al variare del computer, della versione del BASIC o delle opzioni di compilazione.
Per ottenere una pausa di durata precisa e indipendente dalla velocità di calcolo del computer si può usare il ciclo DO...LOOP inserito nel seguente sottoprogramma ciclorit, al quale il programma principale passa il numero di secondi voluti per il ritardo (temporit).

```
REM PROGRAMMA PRINCIPALE
.
CALL ciclorit(10)
.
END

SUB ciclorit (temporit) STATIC
 finecicl = TIMER + temporit
 DO WHILE TIMER < finecicl
 LOOP
END SUB
```

In questo sottoprogramma, TIMER è una funzione incorporata (→ pag. 54) che fornisce il numero di secondi trascorsi dalla mezzanotte fino al momento attuale (che è l’inizio della pausa), e finecicl è il numero di secondi, sempre a partire dalla mezzanotte, dopo il quale terminerà la pausa (vedi lo schema di Figura 26). Il sottoprogramma ciclorit viene eseguito fin tanto che TIMER risulta minore di finecicl.

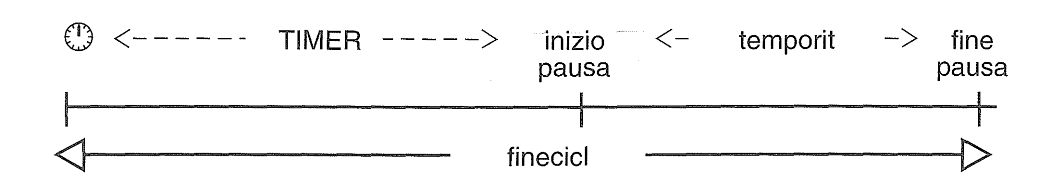


Figura 26 *- Primo schema temporale del sottoprogramma ciclorit*

In realtà il sottoprogramma dovrebbe prevedere anche il caso che la pausa inizi prima di mezzanotte e finisca dopo mezzanotte, come indica lo schema di Figura 27.

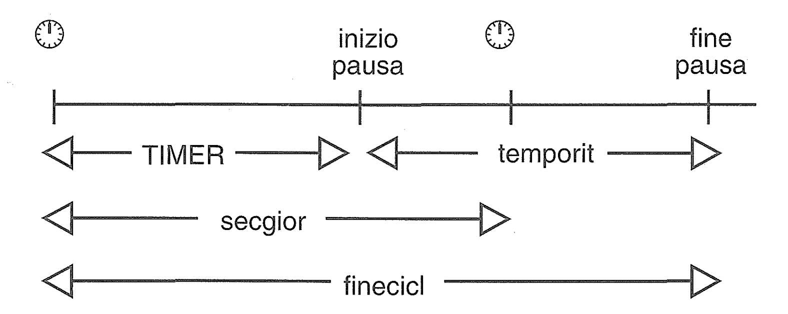


Figura 27 - Secondo schema temporale del sottoprogramma ciclorit

In questo caso finecicl risulta maggiore del numero di secondi contenuti in un giorno (calcolato nella costante secgior), e va diminuito di tale numero; in tal modo rappresenta la sola parte della pausa che va da mezzanotte alla fine, e che viene realizzata come nel sottoprogramma già visto. La parte della pausa che va dall'inizio a mezzanotte viene realizzata ripetendo un ciclo fino a che TIMER ha un valore positivo, cioè fino a mezzanotte, quando viene reimpostato a zero.
Il sottoprogramma completo risulta pertanto il seguente.

```
SUB ciclorit (temporit) STATIC
 CONST secgior = 24& * 60& * 60&
 finecicl = TIMER + temporit
 IF finecicl > secgior THEN
  finecicl = finecicl - secgior
  DO WHILE TIMER > finecicl
  LOOP
 END IF
 DO WHILE TIMER < finecicl
 LOOP
END SUB
```

PROCEDURE RICORSIVE

A differenza delle precedenti versioni, QBASIC permette di scrivere **procedure ricorsive,** cioè programmi che chiamino se stessi, oppure che chiamino altre procedure che a loro volta chiamano i programmi di partenza.
Un modo semplice per illustrare il concetto di ricorsività è di considerare il fattoriale di un numero intero *n*, che si indica con il simbolo *n*! e si può definire con la seguente formula:

$n! = n*(n-1)*(n-2)*...*2*1$

Ad esempio, 5! si calcola come segue:

$5!=5*4*3*2*1 = 120$

Ma il fattoriale di un numero intero si può definire anche con la seguente formula ricorsiva:

```
n! = n*(n-1)!
```

In questo caso, 5! si calcolerebbe come segue:

```
5! = 5*4!
4! = 4*3!
3! = 3*2!
2! = 2*1!
1! = 1*0!
```

A questo punto, 0! deve essere definito e, per ottenere un risultato uguale a quello fornito dalla definizione precedente, si deve porre 0! = 1. La precedente formula ricorsiva può essere inserita nella funzione Fattor#, chiamata e definita dal programma che segue.

```
DECLARE FUNCTION Fattor# (N%)
Formato$ = "###_! = ####################"
DO
 INPUT "Scrivi un numero da 0 a 20
 (o -1 per finire): ", Num%
 IF Num% >= 0 AND Num% <= 20 THEN
  PRINT USING Formato$; Num%; Fattor#(Num%)
 END IF
LOOP WHILE Num% > 0
END

FUNCTION Fattor# (N%) STATIC
 IF N% > 0 THEN
  Fattor# = N% * Fattor#(N% - 1)
 ELSE
  Fattor# = 1
 END IF
END FUNCTION
```

Osserviamo che, sebbene la precedente procedura ricorsiva impieghi variabili statiche (essendo questa la modalità preimpostata), è in generale meglio usare variabili automatiche. In tal modo non ci si deve preoccupare del fatto che le chiamate ricorsive della procedura sovrascrivano i valori delle variabili da una chiamata precedente.

Capitolo 12

File di programma e di dati



FILE DI PROGRAMMA: SALVATAGGIO

I programmi QBASIC visti finora risiedono fisicamente nella memoria centrale (o RAM) del computer, dalla quale vengono cancellati al suo spegnimento. Se invece si vogliono memorizzare in modo permanente per un utilizzo futuro, devono essere registrati su un disco (hard o floppy), diventando in tal modo dei **file di programma**. Dal punto di vista fisico, un file è quindi una particolare zona di un disco magnetico, dove sono memorizzate informazioni in modo permanente ma modificabile.
La procedura per registrare (o "salvare") un file di programma è la seguente:

- si attiva la prima riga dello schermo di QBASIC (con il tasto <Alt>)
- si apre la finestra File (con il tasto <Invio> o con <F>)
- si sceglie una delle voci Salva, Salva con nome o Esci
- se si è scelto Salva o Salva con nome, si scrive il nome che si vuole dare al file. La sessione di lavoro con QBASIC continua presentando il file che si è appena salvato
- se si è scelto Esci, si risponde <Sì> alla domanda

```
Il file non è stato salvato. Salvare?
```

quindi si scrive il nome che si vuole dare al file. In tal modo la sessione di lavoro con QBASIC termina. Se si vuole salvare un file di programma sul floppy disc presente nel drive A, bisogna far precedere il nome del file dai caratteri A: (scrivendo, ad esempio, A:PROVA).

FILE DI PROGRAMMA: APERTURA

La procedura per richiamare (o "aprire") un file di programma salvato in precedenza è la seguente:

- si attiva la prima riga dello schermo di QBASIC (con il tasto <Alt>)
- si apre la finestra File (con il tasto <Invio> o con <F>)
- si sceglie la voce Apri
- si scrive il nome del file da aprire (non occorre l'estensione .BAS), oppure lo si sceglie nella lista proposta a video (per "entrare" nella finestra si preme il tasto del tabulatore).

Il file così aperto può essere eseguito o modificato, e in tal caso essere salvato con o senza le modifiche apportate.

FILE DI DATI

Oltre ai file di programma, QBASIC permette anche di utilizzare **file di dati**, costituiti non già da parole chiave convenzionali, ma da informazioni scritte dall'utente secondo la sintassi e il formato a lui più congeniali.
Dal punto di vista logico, un file di dati è costituito da uno o più blocchi di informazioni detti **record** relative a diversi oggetti o individui; questi potrebbero essere, ad esempio, i libri di una biblioteca o gli impiegati di un'azienda. Il record di ogni individuo è suddiviso a sua volta in un certo numero di **campi**, ciascuno dei quali contiene i diversi

tipi di informazioni relative a quell'individuo.
L'impiego di file di dati in programmi QBASIC semplifica grandemente i seguenti compiti:

- creare, manipolare e memorizzare grandi quantità di dati
- accedere a diversi insiemi di dati in un programma
- usare lo stesso insieme di dati in diversi programmi.

FILE SEQUENZIALI E AD ACCESSO CASUALE

QBASIC permette di utilizzare due tipi di file dati:

- file (ad accesso) **sequenziale**, nei quali i record dei diversi individui possono avere lunghezze diverse (vedi Figura 28)

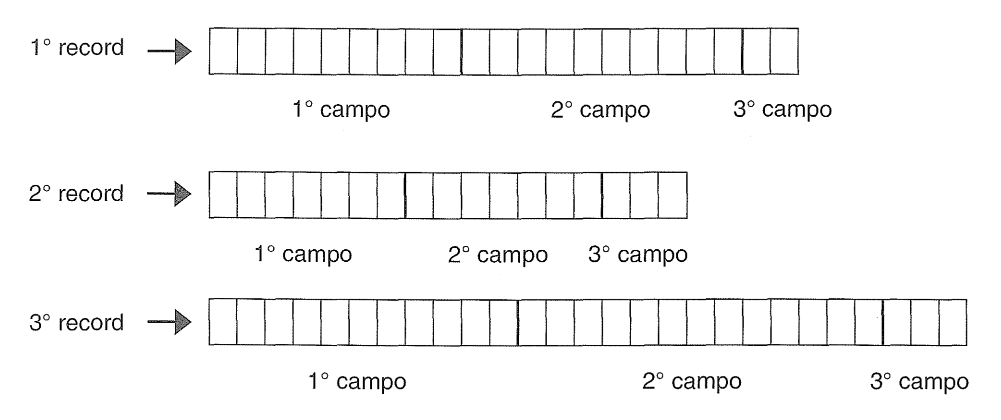


Figura 28 - Record e campi in un file sequenziale

- file (ad accesso) **casuale,** nei quali i record hanno tutti la stessa lunghezza (vedi Figura 28).

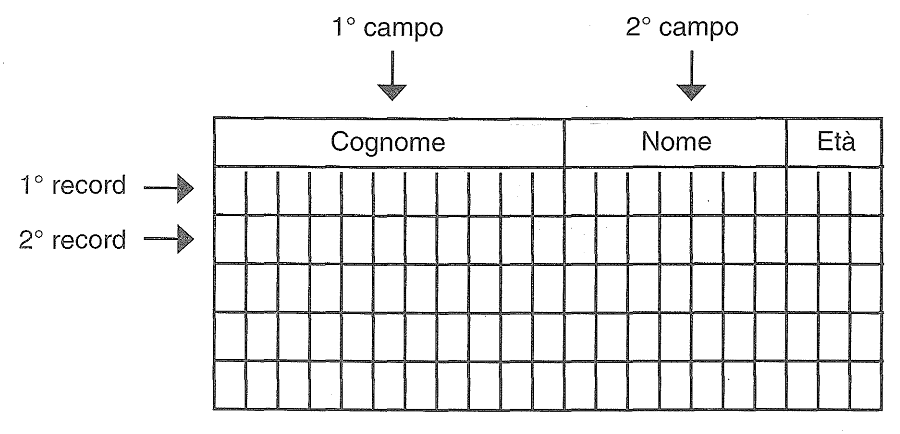


Figura 29 - Record e campi in un file casuale

La diversa denominazione ha origine dalle diverse modalità di accesso ai dati. Un file sequenziale può essere paragonato a una cassetta audio, sulla quale siano registrati vari brani musicali: dato che essi possono essere di lunghezze differenti, l'unico modo per trovare un particolare brano è di far scorrere in sequenza tutto il nastro che lo precede. In modo analogo, per accedere ai dati del record numero 100, è necessario leggere tutti i record compresi fra il numero 1 e il 99, impiegando quindi un tempo (detto *tempo di accesso*) piuttosto lungo.
Invece un file ad accesso casuale è simile a un disco LP o CD, nel quale è possibile posizionare direttamente la testina di lettura sul brano desiderato, senza dover ascoltare tutti quelli che lo precedono. La stessa cosa succede con un file casuale: dato che tutti i suoi record hanno la stessa lunghezza, è possibile accedere a un record qualsiasi semplicemente indicandone il numero, e quindi impiegando un tempo di accesso più breve rispetto a un file sequenziale.
A parità di dati contenuti, i file sequenziali occupano uno spazio minore di quelli casuali e, come vedremo, vengono trattati in modo diverso. Inoltre, devono essere letti sempre per intero, anche se si ha bisogno di conoscere un solo dato memorizzato, mentre la modifica di un dato richiede la riscrittura dell'intero file; per queste ragioni sono adatti per contenere informazioni soggette ad aggiornamenti poco frequenti.
Invece i file ad accesso casuale, grazie alla loro organizzazione più regolare, permettono l'accesso (per la lettura o la modificazione) al singolo dato, e sono quindi particolarmente adatti per contenere informazioni soggette ad aggiornamenti frequenti.

SALVATAGGIO E APERTURA (OPEN)

A differenza dei file di programma, i file dati vengono salvati e aperti non scegliendo dei comandi presentati a video, ma eseguendo semplici programmi QBASIC, costituiti da opportune istruzioni. Ciò è necessario in quanto i dati che costituiscono un record, prima di venire scritti su disco, sono parcheggiati (in gergo si dice *allocati*) in una particolare zona di memoria detta *memoria di transito* (o *buffer*) del file; qui essi vengono organizzati per il successivo invio al disco. In maniera analoga, non è possibile leggere direttamente i file dati presenti su un disco, ma solo dopo averli trasferiti - ed eventualmente organizzati - nella memoria di transito.
Prima di eseguire su un file dati un'operazione di scrittura (detta OUTPUT), di lettura (INPUT) o di aggiunta (APPEND), è necessario aprire il file con un'istruzione OPEN.
L'istruzione OPEN contiene, come vedremo nei paragrafi successivi, l'indicazione del tipo di operazione che si vuole eseguire sul file. Essa inoltre associa al file un numero intero compreso fra 1 e 255, che viene usato come riferimento abbreviato nelle successive operazioni di lettura/scrittura sul file.

CHIUSURA (CLOSE, RUN)

Dopo avere aperto un file, è necessario chiuderlo con l'istruzione

`CLOSE [`*`numero di file`*`]`

Questa ha due effetti: scrive nel file tutti i dati presenti nel buffer e libera il *numero di*

file associato al file, che può così essere usato in un'altra istruzione OPEN. Usata senza numero di file come argomento, l'istruzione CLOSE chiude tutti i file aperti.
Un file dati viene chiuso, oltre che con l'istruzione CLOSE, anche se il programma che esegue l'operazione di ingresso/uscita ha termine, oppure trasferisce il controllo a un altro programma, con l'istruzione

```
RUN PROGRAMMA
```

Questa viene scritta di solito come ultima istruzione di un programma, se si vuole che al termine sia eseguito il programma indicato (anch'esso con estensione .BAS).

FILE SEQUENZIALI: CREAZIONE (WRITE #)

Un programma che crea un file sequenziale di nome Misure.dat (o, se già esiste, lo apre) e vi scrive dei record costituiti dai campi Nome, Cod. fisc., Altezza, Età può essere il seguente:

```
REM ***SCRIVESEQ***
OPEN "Misure.dat" FOR APPEND AS #1
DO
 INPUT "Nome (o <Invio> per terminare): ", Nome$
 IF Nome$ <> "" THEN
  INPUT "Cod. fisc.: ", Cf$
  INPUT "Altezza: ", Altezza
  INPUT "Età: ", Eta
  WRITE #1, Nome$, Cf$, Altezza, Eta
 END IF
LOOP UNTIL Nome$ = ""
CLOSE #1
END
```

In esso l'istruzione

```
OPEN "Misure.dat" FOR APPEND AS #1
```

crea (o apre) un file dati di nome Misure.dat nella modalità di scrittura (o di aggiunta) di dati nei suoi record.
Il ciclo

```
DO
...
LOOP UNTIL Nome$ = ""
```

permette di scrivere quanti record si vogliono, e termina quando si preme il tasto <Invio> invece di scrivere un Nome.
Le varie istruzioni

```
INPUT
```

assegnano i dati immessi da tastiera a variabili di stringa o numeriche, i cui contenuti sono poi copiati nei campi del file dall'istruzione

```
WRITE #1
```

Quando l'istruzione WRITE # scrive i dati nei campi, delimita ogni variabile di stringa con le virgolette ("), separa i campi con la virgola (,) e pone alla fine di ciascun record i caratteri di ritorno carrello (CR) e avanzamento linea (LF), che corrispondono alla pressione del tasto <Invio>, come è indicato in Figura 30.
Naturalmente questi caratteri non sono visualizzati quando si va a leggere il file con uno dei programmi di lettura che esamineremo nei prossimi paragrafi, mentre si possono vedere con l'istruzione LINE INPUT # (esaminata più avanti), con il comando TYPE di MS-DOS o con un qualsiasi programma editore di testo.

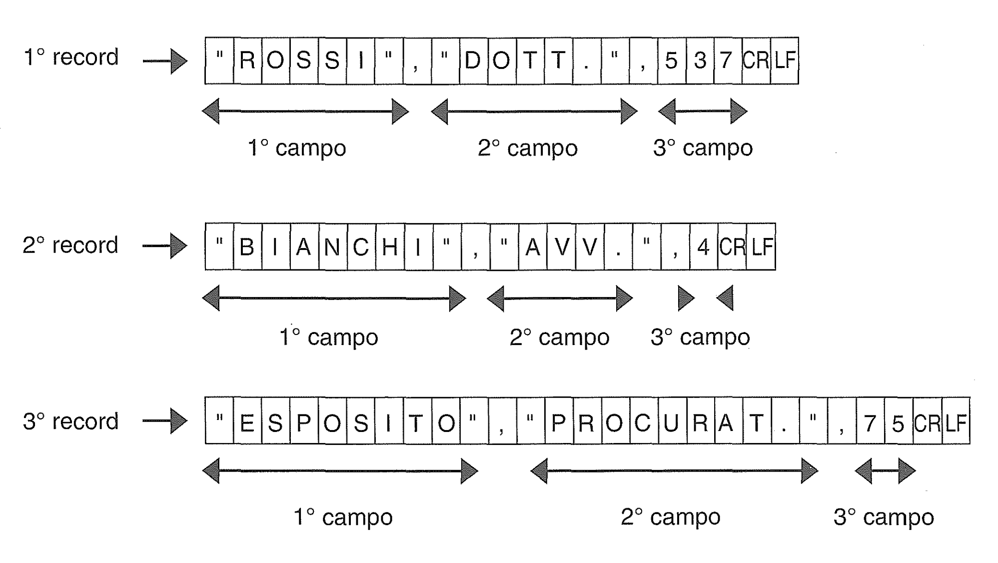


Figura 30 *- Organizzazione dei dati in un file sequenziale*

Apertura per aggiunta (APPEND) e per scrittura (OUTPUT)

Quando si crea un file sequenziale, è anche possibile aprirlo in modalità di scrittura (OUTPUT) invece di aggiunta (APPEND), scrivendo quindi

```
OPEN "Misure.dat" FOR OUTPUT AS #1
```

anziché

```
OPEN "Misure.dat" FOR APPEND AS #1
```

Tuttavia, la modalità di scrittura (OUTPUT) determina la cancellazione di eventuali dati già presenti nel file, per cui è più sicuro e quindi preferibile usare la modalità APPEND tutte le volte che si devono scrivere dei dati, riservando la OUTPUT ai casi (rari) in cui si voglia riscrivere il file.

Nota. L'apertura in modalità di scrittura si può riferire anche a una periferica quale la stampante (che ha come nome o indirizzo LPT1:); quindi un'istruzione del tipo

```
OPEN "LPT1:" FOR OUTPUT AS #1
```

ha l'effetto di inviare alla stampante i dati che saranno elaborati successivamente (vedremo un esempio di utilizzo nel paragrafo "Routine di gestione degli errori, a pag. 186).

FILE SEQUENZIALI: LETTURA (INPUT #, EOF)

Per leggere un file sequenziale si deve scrivere un programma simile a quello usato per la sua creazione, e che in particolare preveda la lettura (INPUT) di tutti i campi dei record in altrettante variabili.
Un programma per leggere il file Misure.dat scritto in precedenza potrebbe essere il seguente:

```
REM ***LEGGESEQ1***
OPEN "Misure.dat" FOR INPUT AS #1
DO UNTIL EOF(1)
 INPUT #1, Nome$, Cf$, Altezza, Eta
 PRINT Nome$, Cf$, Altezza, Eta
LOOP
CLOSE #1
END
```

In esso l'istruzione

```
INPUT #1, Nome$, Cf$, Altezza, Eta
```

legge un record alla volta, nell'ordine di immissione, dal file Misure.dat, e assegna i suoi campi alle variabili Nome$, Cf$, Altezza, Età.
La funzione

```
EOF
```

(End Of File) controlla se l'istruzione INPUT #1 ha letto l'ultimo record, nel qual caso fornisce il valore -1 (vero) e il ciclo di richiesta dati ha termine; in caso contrario EOF fornisce il valore 0 (falso) e viene letto il record successivo del file.
Il programma precedente può essere leggermente modificato come segue, se si vogliono visualizzare i soli record con età inferiore a un determinato valore:

```
REM ***LEGGESEQ2***
OPEN "Misure.dat" FOR INPUT AS #1
INPUT "Lettura dei record con età inferiore a"; Emax
DO UNTIL EOF(1)
 INPUT #1, Nome$, Peso$, Altezza$, Eta
 IF Eta < Emax
  PRINT Nome$, Peso$, Altezza$, Eta
 END IF
LOOP
CLOSE #1
END
```

ALTRE POSSIBILITÀ DI LETTURA

Finora abbiamo usato l'istruzione INPUT # per leggere un record (cioè tutti i caratteri che precedono una sequenza CR-LF) e assegnare i suoi diversi campi alle variabili elencate dopo INPUT #. Vi sono altre due possibilità di leggere dati da file sequenziali, sia come record (LINE INPUT #) sia come sequenze di byte non formattate (INPUT$).

Istruzione LINE INPUT

L'istruzione LINE INPUT # permette a un programma di leggere un record (o linea di testo) esattamente come è scritto in un file, senza interpretare le virgolette e le virgole come delimitatori di campo. Ciò è particolarmente utile in programmi che lavorano con file di testo ASCII. LINE INPUT # legge un'intera linea di un file sequenziale in una singola variabile di stringa. Il seguente programma legge ogni linea del file Misure, quindi la visualizza sullo schermo.

```
REM ***LEGGESEQ3***
OPEN "Misure.dat" FOR INPUT AS #1
DO UNTIL EOF(1)
 LINE INPUT #1, Temp$
 PRINT Temp$
LOOP
```

Funzione INPUT$

Un altro modo per leggere dati da un file (sequenziale, ma anche di altro tipo), è di usare la funzione INPUT$. A differenza di INPUT # e di LINE INPUT #, che leggono una linea per volta da un file sequenziale, INPUT$ legge un numero specificato di caratteri da un file, secondo la sintassi della seguente istruzione:

```
T$ = INPUT$(50, #1)
```

che legge 50 caratteri dal file numero 1, e li assegna alla variabile T$.

Un esempio di utilizzo della funzione INPUT$ è costituito dal programma seguente, che chiede il nome di un file da aprire e lo copia sullo schermo, visualizzando solo i caratteri alfanumerici e di interpunzione (quelli cioè compresi fra il carattere ASCII 32 e il 126).

```
REM ***LEGGESEQ4***
INPUT "Nome del file da aprire: ", F$
IF F$ = "" THEN END
OPEN F$ FOR INPUT AS #1
DO UNTIL EOF(1)
 T$ = INPUT$(1, #1)
 Tn = ASC(T$)
 SELECT CASE Tn
  CASE 32 TO 126
   PRINT T$;
  CASE ELSE
 END SELECT
LOOP
```

Capitolo 13

File ad accesso casuale

ORGANIZZAZIONE DEI RECORD

In un file dati ad accesso casuale i record sono memorizzati in modo differente rispetto a un file sequenziale. Dato che ogni record e ogni campo hanno una lunghezza fissa, queste lunghezze determinano il punto d'inizio e di fine di ogni record o campo, senza bisogno di usare virgole per separare i campi, né sequenze CR-LF per separare i record. L'**organizzazione dei record** in un file ad accesso casuale è indicata in Figura 31.

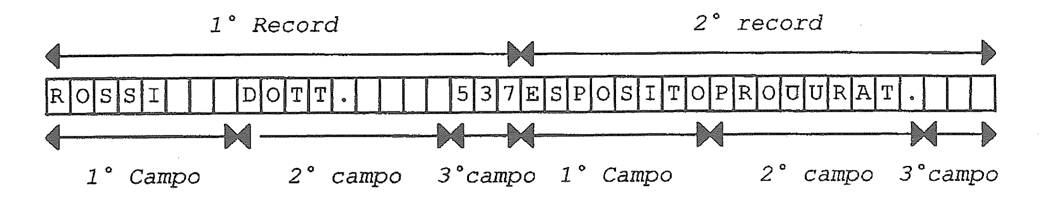


Figura 31 - Organizzazione dei record in un file ad accesso casuale

AGGIUNTA DI DATI

Un programma che aggiunga dati a un file ad accesso casuale è costituito dalle tre parti seguenti:

- definizione dei campi del record
- apertura del file nella modalità ad accesso casuale e indicazione della lunghezza del record
- immissione dei dati nel record e sua memorizzazione.

Esaminiamo separatamente ciascuna parte.

DEFINIZIONE DEI CAMPI DEL RECORD (TYPE...END TYPE)

Il record viene definito con l'istruzione TYPE...END TYPE, che permette di creare un tipo di dati composito, cioè costituito da stringhe ed elementi numerici. Ad esempio, per definire

- Cognome come stringa lunga 15 caratteri
- Nome come stringa lunga 10
- Età come intero
- Stipendio come singola precisione

si può scrivere:

```
TYPE Tiprec
 Cognome AS STRING * 15
 Nome AS STRING * 10
 Eta AS INTEGER
 Stipendio AS SINGLE
END TYPE
```

Segue la dichiarazione di una variabile (in questo caso Varrec) del tipo Tiprec appena definito con l'istruzione:

```
DIM Varrec AS Tiprec
```

APERTURA DEL FILE E INDICAZIONE DELLA LUNGHEZZA DEL RECORD (LEN =)

A differenza dei file sequenziali, nei file ad accesso casuale non c'è più differenza tra apertura per lettura e per scrittura. La lunghezza del record ad accesso casuale, che ha come valore preimpostato 128 byte, dev'essere uguale a quella della variabile Varrec. Ciò si ottiene con la clausola LEN = nell'istruzione OPEN. Continuando con l'esempio precedente, per aprire un file di nome Organico.dat si scriverebbe:

```
OPEN "Organico.dat" FOR RANDOM AS #1 LEN = LEN(Varrec)
```

Osserviamo che la lunghezza della variabile Varrec calcolata dalla funzione LEN è 31 byte, derivanti dalla somma dei 15 di Cognome, più i 10 di Nome, più i 2 di Eta (trattandosi di una variabile di tipo Intero, vedi paragrafo "Tipi di variabili" a pag. 20), più i 4 di Salario (che è di tipo Singola precisione).

IMMISSIONE DEI DATI NEL RECORD E SUA MEMORIZZAZIONE (PUT #)

L'immissione dei dati nei vari campi del record può avvenire con istruzioni di questo tipo:

```
INPUT "Cognome"; Varrec.Cognome
INPUT "Nome"; Varrec.Nome
INPUT "Eta"; Varrec.Eta
INPUT "Stipendio"; Varrec.Stipendio
```

alle quali segue l'istruzione che memorizza i dati di Varrec nel record:

```
PUT #1,  , Varrec
```

Il programma completo risulta quindi SCRIVECAS di Figura 32, che impiega tutte variabili di tipo intero:

```
REM ***SCRIVECAS***
DEFINT A-Z
TYPE Tiprec
 Cognome AS STRING * 15
 Nome AS STRING * 10
 Eta AS INTEGER
 Stipendio AS SINGLE
END TYPE
DIM Varrec AS Tiprec
OPEN "Organico.dat" FOR RANDOM AS #1 LEN =
LEN(Varrec)
Numrec = LOF(1) \ LEN(Varrec)
DO
 INPUT "Cognome"; Varrec.Cognome
 INPUT "Nome"; Varrec.Nome
 INPUT "Eta"; Varrec.Eta
 INPUT "Stipendio"; Varrec.Stipendio
 Numrec = Numrec + 1
 PUT #1, Numrec, Varrec
 INPUT "Si vuole continuare? (S/N) ", R$
LOOP UNTIL UCASE$(R$) = "N"
CLOSE #1
END
```

***Figura 32** - Programma di aggiunta dati a un file casuale*

Se il file Organico.dat già esiste, il programma SCRIVECAS gli aggiunge record alla fine senza sovrascrivere quelli esistenti. Ciò è reso possibile dall'istruzione

```
Numrec = LOF(1) \ LEN(Varrec)
```

che esegue le tre operazioni seguenti:

- La funzione LOF(1) calcola il numero totale di byte del file Organico.dat, fornendo 0 se il file è nuovo o è vuoto;
- La funzione LEN(Varrec) calcola il numero di byte del record (Varrec è definita della stessa struttura del tipo Tiprec definito dall'utente);
- Il numero di record del file è quindi calcolato dalla formula

$$\text{numero di record} = \frac{\text{byte totali del file}}{\text{byte di un record}}$$

L'ultima operazione è possibile in quanto tutti i record del file ad accesso casuale hanno la stessa lunghezza, mentre non sarebbe possibile in un file sequenziale.

AGGIUNTA DI DATI NELLE PRECEDENTI VERSIONI DI BASIC (FIELD, MK...)

L'istruzione TYPE...END TYPE costituisce una grande semplificazione rispetto alle precedenti versioni di BASIC; in esse la definizione del record avviene con l'istruzione FIELD (peraltro mantenuta in QBASIC, per una ragione di compatibilità), che accetta però solo variabili di stringa (anche per i dati numerici). Quindi il record dell'esempio precedente andrebbe definito con la seguente istruzione:

```
FIELD #1, 15 AS Cognome$, 10 AS Nome$, 2 AS Eta$, 4 AS Stipe$
```

A questo punto i dati immessi dall'utente vanno assegnati dapprima a variabili temporanee, con istruzioni del tipo

```
INPUT "Cognome "; C$
INPUT "Nome "; N$
INPUT "Età "; Eta
INPUT "Stipendio "; Stipe
```

quindi vanno "spostati" in una particolare memoria di transito *(buffer)* con istruzioni del tipo

```
LSET Cognome$=C$
LSET Nome$=N$
```

per le variabili di stringa, e del tipo

```
LSET Eta$=MKI$(Eta)
LSET Stipe$=MKL$(Stipe)
```

per le variabili numeriche. Nelle ultime due istruzioni sono state impiegate le funzioni MKI$, MKL$ che convertono i numeri in un formato di stringa adatto per l'uso in un'istruzione FIELD.
Le funzioni di questo tipo sono in tutto quattro, e sono indicate in Tabella 19.

Funzione	Tipo dell'argomento	Lunghezza della stringa fornita
MKI$	Intero	2 byte
MKL$	Intero LONG	4 byte
MKS$	Singola precisione	4 byte
MKD$	Doppia precisione	8 byte

Tabella 19 *- Argomenti e risultati delle funzioni di conversione MKI$, MKL$, MKS$, MKD$*

LETTURA GLOBALE DI DATI

Il seguente programma legge tutti i record del file Organico.dat in modo globale, dal primo all'ultimo.

```
TYPE Tiprec
 Cognome AS STRING * 15
 Nome AS STRING * 10
 Eta AS INTEGER
 Stipendio AS SINGLE
END TYPE
DIM Varrec AS Tiprec
OPEN "Organico.dat" FOR RANDOM AS #1 LEN = LEN(Varrec)
Numrec = LOF(1) \ LEN(Varrec)
FOR Nr = 1 to Numrec
 GET #1, Nr, Varrec
 PRINT "Cognome: ", Varrec.Cognome
 PRINT "Nome: ", Varrec.Nome
 PRINT "Eta: ", Varrec.Eta
 PRINT "Stipendio: ", Varrec.Stipendio
NEXT
CLOSE #1
END
```

In esso l'istruzione GET legge dal file numero 1 (cioè da Organico dat.) i dati del record numero Nr, e li assegna alla variabile Varrec, in modo che possano essere visualizzati dalle successive istruzioni PRINT. La sua forma sintattica è

GET [#] *numfile* [, [*numrecord*] [, *variabile*]]

dove la *variabile* può essere di un tipo qualsiasi (anche di stringa a lunghezza variabile, o di tipo definito dall'utente) e, se viene omessa, i dati letti sono memorizzati in una memoria di transito *(buffer).*

LETTURA DI DATI NELLE PRECEDENTI VERSIONI DI BASIC (CV...)

Anche la lettura di un file casuale è più laboriosa con le precedenti versioni di BASIC, in quanto le stringhe che rappresentano numeri vanno nuovamente convertite in numeri con istruzioni del tipo

```
PRINT CVI(Eta$)
PRINT CVL(Stipe$)
```

Le precedenti funzioni CVI, CVL, che convertono le stringhe dell'istruzione FIELD in numeri, appartengono all'insieme delle quattro funzioni di questo tipo indicate in Tabella 20.

Funzione	Lunghezza dell'argomento	Tipo di numero fornito
CVI CVL CVS CVD	2 byte 4 byte 4 byte 8 byte	Intero Intero LONG Singola precisione Doppia precisione

Tabella 20 *- Argomenti e risultati delle funzioni di conversione CVD, CVI, CVL, CVS*

RICHIAMO DI RECORD TRAMITE I NUMERI DI RECORD

Come ho accennato, uno dei maggiori vantaggi dei file ad accesso casuale rispetto a quelli sequenziali è la possibilità di richiamare un record qualsiasi, indicandone il numero in un'istruzione GET. Il seguente programma LEGGEREC chiede un numero di record e ne visualizza i dati sullo schermo.

```
REM *** LEGGERE C ***
DEFINT A-Z
CONST Falso = 0, Vero = NOT Falso
TYPE Tiprec
 Cognome AS STRING * 15
 Nome AS STRING * 10
 Eta AS INTEGER
 Stipendio AS SINGLE
END TYPE
DIM Varrec AS Tiprec
OPEN "Organico.dat" FOR RANDOM AS #1 LEN = LEN(Varrec)
Numrec = LOF(1) \ LEN(Varrec)
Ancora = Vero
DO
 PRINT "Indica il numero di record ";
 PRINT "(0 per finire): ";
 INPUT "", Nr
 IF Nr > 0 AND Nr < Numrec THEN
  GET #1, Nr, Varrec
 ELSEIF Nr = 0 THEN
  Ancora = Falso
 ELSE
  PRINT "Valore non valido."
 END IF
LOOP WHILE Ancora
END
```

LETTURA/SCRITTURA BINARIA DI FILE

L'accesso binario è la terza modalità per leggere o scrivere i dati di un file, oltre all'accesso sequenziale e a quello casuale. Esso permette di accedere in modo grezzo ai byte di qualsiasi file, quindi non solo di quelli in formato ASCII, ma anche in formato Microsoft Word o di tipo eseguibile (cioè con estensione .EXE).
I file aperti in accesso binario sono trattati come sequenze non formattate di byte; sebbene sia possibile leggere o scrivere un record in un file aperto in modalità binaria, non è necessario che il file sia organizzato in record di lunghezza fissa, dato che l'accesso binario non tratta record ma caratteri (che si possono considerare come record). Un programma che legge il file Misure.dat in modalità binaria è il seguente LEGGEBIN:

```
REM *** LEGGEBIN ***
OPEN "Misure.dat" FOR BINARY AS #1
DO UNTIL EOF(1)
 GET #1, , i%
 PRINT i%
LOOP
```

In esso la prima istruzione apre il file Misure.dat in modalità binaria e gli assegna il numero 1. L'istruzione GET ha la stessa forma sintattica vista nel paragrafo "Lettura globale di dati" (→ pag. 141), però in questo caso il parametro *numrecord* indica la posizione del byte in cui comincia la lettura, riferita all'inizio del file. Se *numrecord* non è indicato, si assume la posizione del carattere successivo all'ultimo letto.
La lunghezza della *variabile* indica il numero di byte che verranno letti, ricordando che una variabile di tipo Intero è lunga due caratteri, una di tipo Intero LONG o in singola precisione quattro caratteri, e una in Doppia precisione otto caratteri (vedi paragrafo "Tipi di variabili" a pag. 20).
Per comprendere il tipo di risultati che si ottengono con la lettura binaria, supponiamo che il file Misure .dat abbia il contenuto "angelo". Questi caratteri vengono memorizzati nel file attraverso i rispettivi codici ASCII esadecimali (vedi paragrafo "Tabella dei codici ASCII" a pag. 14), riportati in Tabella 21.

Carattere	Codice ASCII esadecimale
"	22
a	61
n	6E
g	67
e	65
l	6C
o	6F

***Tabella 21** - Codici ASCII esadecimali di caratteri memorizzati nel file Misure.dat*

Quando l'istruzione `GET #1, , i%` legge la successione dei numeri esadecimali

22 61 6E 67 65 6C 6F 22

presente nel file, ne inserisce due alla volta nella variabile i% scambiandoli di ordine e convertendoli nel sistema di numerazione decimale (un modo semplice per eseguire la conversione da numeri esadecimali è di usare la Calcolatrice, nella visualizzazione scientifica, presente negli Accessori di Windows). Perciò la variabile i% assumerà successivamente i valori decimali indicati nella prima colonna di Tabella 22.

Valore decimale	**Valore esadecimale**
24866	61 22
26478	67 6E
27749	6C 65
8815	22 6F

Tabella 22 *- Equivalenti decimali di alcuni numeri esadecimali*

ALTRE OPERAZIONI CON I FILE (NAME, KILL, CHDIR, MKDIR, FILES)

QBASIC possiede diverse istruzioni che permettono di eseguire da programma alcune operazioni su file e sottoindirizzari, operazioni che di solito vengono eseguite attraverso comandi del sistema operativo MS-DOS. Anche se queste istruzioni possono essere eseguite in modo immediato, raramente lo sono, dato che QBASIC riporta automaticamente al sistema operativo MS-DOS dopo l'esecuzione di un programma.
La Tabella 23 permette di confrontare i nomi dei comandi DOS e delle istruzioni QBASIC che eseguono tali operazioni.

operazione	**comando DOS**	**istruzione QBASIC**
cambio di nome a un file	REN[AME] CEE UE	NAME "CEE" AS "UE"
cancellazione di un file da disco	ERASE agenda. bas DEL agenda. bas	KILL "agenda. bas"
cambio dell'indirizzario corrente	CHDIR C:DBASE CD C:DBASE	CHDIR "C:DBASE"
creazione di un nuovo indirizzario	MKDIR C:TEMP MD C:TEMP	MKDIR "C:TEMP"
cancellazione di un indirizzario (vuoto)	RMDIR C:TEMP RD C:TEMP	RMDIR "C:TEMP"
visualizzazione dei file dell'indirizzario corrente, o di quello specificato	DIR DIR A: DIR *.BAS	FILES FILES "A" FILES "*.BAS"

Tabella 23 *- Nomi dei comandi DOS e delle istruzioni QBASIC che eseguono le stesse operazioni*

Capitolo 14

La grafica



SCHEDE GRAFICHE

QBASIC dispone di istruzioni specifiche per disegnare grafici sullo schermo o trasferirli sulla stampante, istruzioni che richiedono l'esistenza di un adattatore grafico all'interno del computer e di uno schermo grafico a esso collegato. Gli adattatori grafici disponibili comprendono le seguenti **schede grafiche:**

HGA (Hercules Graphics Adapter)
CGA (Color Graphics Adapter)
EGA (Enhanced Graphics Adapter)
VGA (Video Graphics Adapter)
MCGA (Memory Controller Gate Array)

Esse differiscono per il *numero di colori* che permettono di visualizzare contemporaneamente e per la massima *risoluzione* che rendono disponibile.

RISOLUZIONE E PIXEL

Già sappiamo che lo schermo di un personal computer permette di visualizzare caratteri alfanumerici o semigrafici disponendoli in una matrice di caselle costituita da 25 righe (orizzontali) e 80 colonne (verticali); ciò si esprime anche dicendo che la risoluzione in modalità testo è 25 righe per 80 colonne.
In realtà ogni carattere visualizzato sullo schermo è ricavato a sua volta da una matrice di 8 righe e 8 colonne (ma sono possibili anche i valori 8*14, 9*14 e 9*16), i cui incroci sono detti **pixel** e risultano accesi o spenti, a seconda del carattere da visualizzare (vedi Figura 33).

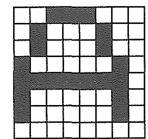


Figura 33 *- Visualizzazione del carattere "A" con una matrice di 8 righe e 8 colonne*

Mentre in modalità testo l'utente può scegliere solo il carattere da visualizzare in una casella, senza intervenire sulle singole configurazioni di pixel, in modalità grafica può accedere direttamente a ogni singolo pixel, disponendo quindi di una risoluzione più elevata.
Se infatti moltiplichiamo le 80 colonne e le 25 righe di testo per gli 8 pixel di cui sono costituite, otteniamo una risoluzione grafica di 640 colonne e 200 righe di pixel, valore anche superato da molte schede grafiche. Naturalmente, al crescere dei numeri di righe e di colonne diminuisce la dimensione dei pixel, avendo così immagini più nitide.
Il pixel, che è quindi il più piccolo elemento di immagine di cui è costituito un oggetto grafico, è individuato dal numero della colonna e della riga che si incrociano in esso, detti *coordinate*.
Le colonne sono numerate da sinistra a destra (coordinata x) e le righe dall'alto in basso (coordinata y) a partire dalla posizione più in alto e a sinistra, che ha coordinate (0,0) ed è detta *home* (vedi Figura 34).

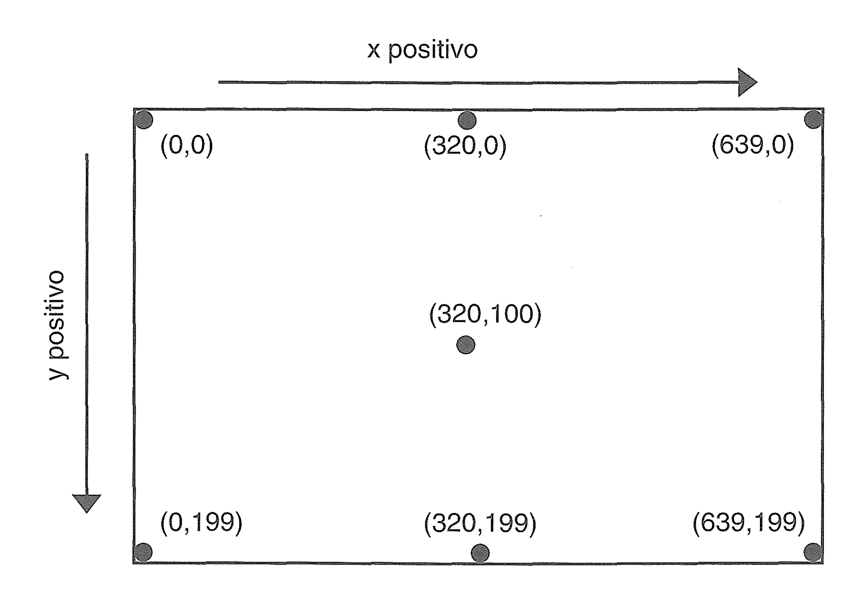


Figura 34 *- Coordinate di particolari pixel nella modalità SCREEN 2*

ISTRUZIONE SCREEN

L'istruzione necessaria per impostare la modalità grafica di schermo è SCREEN, che nel caso più semplice ha la seguente forma sintattica:

```
SCREEN n
```

dove *n* è un numero intero che indica il modo operativo prescelto, secondo quanto indicato di seguito (tra parentesi, dopo il modo operativo, sono riportate le schede grafiche supportate).

0 (CGA, EGA, VGA, MCGA, HGA)
È il valore preimpostato, e corrisponde alla modalità di testo con risoluzione di 80 colonne e 25 righe (ma sono possibili anche i valori 40*25, 40*43, 40*50, 80*43 e 80*50). Supporta fino a 16 colori scelti da 64 con schede EGA o VGA, e fino a un massimo di 8 pagine di memoria video.

1 (CGA, EGA, VGA, MCGA)
Risoluzione grafica 320*200, di testo 40*25 e casella carattere con risoluzione 8*8. Supporta 4 colori di primo piano, 16 di sfondo e 1 pagina di memoria video.

2 (CGA, EGA, VGA, MCGA)
Risoluzione grafica 640*200, di testo 80*25 e casella carattere con risoluzione 8*8. Supporta 1 colore di primo piano, 1 di sfondo e 1 pagina di memoria video.

3 (HGA, Olivetti, AT&T)

Risoluzione grafica 720*348, di testo 80*25 e casella carattere con risoluzione 9*14. Supporta 2 pagine di memoria video. Prima di poterla usare si deve richiamare il driver Hercules MSHERC.COM.

4 (Olivetti, AT&T)
Risoluzione grafica 640*400, di testo 80*25 e casella carattere con risoluzione 8*16. Supporta 1 colore scelto da 16 come colore di primo piano, e 1 pagina di memoria video.

7 (EGA, VGA)
Risoluzione grafica 320*200, di testo 40*25 e casella carattere con risoluzione 8*8. Supporta fino a 16 colori, e 8 pagine di memoria video se la scheda EGA ha una memoria RAM superiore a 64 kbyte, altrimenti 2 pagine di memoria video.

8 (EGA, VGA)
Risoluzione grafica 640*200, di testo 80*25 e casella carattere con risoluzione 8*8. Supporta fino a 16 colori, e 4 pagine di memoria video se la scheda EGA ha una RAM superiore a 64 kbyte, altrimenti 1 pagina di memoria video.

9 (EGA, VGA)
Risoluzione grafica 640*350, di testo 80*25 o 80*43 e casella carattere con risoluzione 8*14 o 8*8. Supporta 16 colori scelti da 64 e 2 pagine di memoria video se l'adattatore ha una memoria RAM superiore a 64 kbyte, altrimenti 4 colori scelti da 16 e 1 pagina di memoria video.

10 (EGA, VGA)
Risoluzione grafica monocromatica 640*350, di testo 80*25 o 80*43 e casella carattere con risoluzione 8*14 o 8*8. Supporta 4 tonalità di grigio scelte da 9 e 2 pagine di memoria video. La scheda deve avere una memoria RAM di 256 kbyte.

11 (VGA, MCGA)
Risoluzione grafica 640*480, di testo 80*30 o 80*60 e casella carattere con risoluzione 8*16 o 8*8. Supporta 2 colori scelti da 262.144 e 1 pagina di memoria video.

12 (VGA)
Risoluzione grafica 640*480, di testo 80*30 o 80*60 e casella carattere con risoluzione 8*16 o 8*8. Supporta 16 colori scelti da 262.144 e 1 pagina di memoria video.

13 (VGA, MCGA)
Risoluzione grafica 320*200, di testo 40*25 e casella carattere con risoluzione 8*8. Supporta 256 colori scelti da 262.144 e una pagina di memoria video.

I valori di risoluzione nelle modalità testo e grafica vanno tenuti presenti per posizionare rispettivamente i caratteri alfanumerici con l'istruzione LOCATE (vista nel paragrafo "Posizionamento del cursore" a pag. 31) e i pixel con le istruzioni grafiche che vedremo in seguito. A questo proposito osserviamo che, mentre l'istruzione LOCATE esprime le coordinate nella forma *riga, colonna,* le istruzioni grafiche le esprimono nella forma *colonna, riga.* Al termine di ogni programma in cui sia stata impostata una modalità grafica di schermo è bene inserire l'istruzione

```
SCREEN 0
```

che riporta al modo di testo standard con risoluzione di 25 righe per 80 colonne.

DISEGNO DI SINGOLI PIXEL (PSET, PRESET)

Il livello più elementare di controllo sulla creazione di disegni è costituito dall'accensione e spegnimento dei singoli pixel. Ciò si ottiene con le istruzioni PSET (per Pixel SET) e PRESET (per Pixel RESET), che hanno le seguenti forme sintattiche:

```
PSET (x, y)
```

che traccia in posizione (*x*, *y*) un pixel con il colore del primo piano, e

```
PRESET (x, y)
```

che traccia in posizione (*x*, *y*) un pixel con il colore dello sfondo (cioè lo spegne).
In uno schermo monocromatico il colore del primo piano è quello impiegato per il testo, cioè di solito bianco o verde chiaro; il colore dello sfondo è nero o verde scuro.
Se si dispone di uno schermo a colori si possono scegliere colori diversi usando PSET e PRESET nelle seguenti forme sintattiche

```
PSET (x, y), colore
PRESET (x, y), colore
```

dove il parametro *colore* ha i valori e gli effetti che vedremo nel paragrafo "Colore di primo piano e di sfondo" (→ pag. 162). Dato che PSET usa come valore preimpostato il colore corrente di primo piano e PRESET quello di sfondo, PRESET impiegato senza colore cancella un punto disegnato con PSET, come mostra l'esempio che segue.

```
SCREEN 2
PRINT "Premi un tasto per terminare"
DO
 FOR X% = 0 TO 640
  PSET (X%, 100)
 NEXT X%
 FOR X% = 640 TO 0 STEP -1
  PRESET (X%, 100)
 NEXT X%
LOOP UNTIL INKEY$ <> ""
END
```

In questo programma l'istruzione `SCREEN 2` imposta la risoluzione grafica 640*200, il primo ciclo `FOR...NEXT` disegna una linea orizzontale da sinistra a destra e il secondo ciclo `FOR...NEXT` la cancella da destra a sinistra.
Sebbene l'istruzione PSET permetta di disegnare qualsiasi figura indirizzando i singoli

pixel, il risultato tende a essere alquanto lento, dato l'elevato numero di pixel che formano in genere una figura. Perciò QBASIC mette a disposizione diverse istruzioni che aumentano notevolmente la velocità con cui vengono disegnate figure quali linee, rettangoli ed ellissi, come vedremo nei paragrafi successivi.

DISEGNO DI SEGMENTI (LINE)

Mentre le istruzioni PSET e PRESET impiegano una sola coppia di coordinate (quelle del punto), l'istruzione LINE impiega due coppie di coordinate per tracciare un segmento, secondo la seguente forma sintattica:

```
LINE (x1, y1)-(x2, y2)[, colore]
```

dove (x_1, y_1) sono le coordinate di un estremo del segmento, (x_2, y_2) quelle dell'altro, e *colore* il solito parametro che vedremo in seguito. Ad esempio, le seguenti istruzioni disegnano un segmento dal pixel di coordinate (10, 10) a quello di coordinate (150, 200):

```
SCREEN 1
LINE (10, 10)-(150, 200)
```

Osserviamo che QBASIC non considera l'ordine delle coppie di coordinate, per cui un segmento da (x_1, y_1) a (x_2, y_2) è lo stesso che un segmento da (x_2, y_2) a (x_1, y_1); quindi la coppia di istruzioni precedenti si può scrivere anche come:

```
SCREEN 1
LINE (150, 200)-(10, 10)
```

L'ordine delle coordinate è tuttavia importante se si impiegano le *coordinate relative,* descritte nel prossimo paragrafo.

COORDINATE RELATIVE (STEP)

Le coordinate impiegate fino a ora erano *coordinate assolute*, in quanto misuravano le distanze orizzontali e verticali (in termini di pixel) contate a partire dalla posizione superiore sinistra dello schermo. Se invece si usa l'opzione STEP in una delle istruzioni grafiche (PSET, PRESET, LINE, CIRCLE, GET, PUT, PAINT), le coordinate vengono riferite all'ultimo punto indirizzato sullo schermo, sono cioè **coordinate relative.** Se, ad esempio, dopo le istruzioni

```
SCREEN 2
LINE (10, 10)-(100, 150)
```

si impartisce l'istruzione

```
PSET STEP (20, 30)
```

il punto sullo schermo verrà disegnato nella posizione di coordinate assolute (120, 180), cioè a distanza orizzontale 20 e a distanza verticale 30 dal punto (100, 150) in cui terminava l'ultimo segmento disegnato.
Notiamo che l'istruzione SCREEN contiene un riferimento implicito al pixel situato nel centro dello schermo, cosicché una successiva opzione STEP avrà effetto da questo punto come risulta dal seguente programma.

```
SCREEN 2
LINE STEP(-310, 100)-STEP(0, -200)
FOR I% = 1 TO 20
 LINE -STEP(30, 0)
 LINE -STEP(0, 5)
NEXT
FOR I% = 1 TO 20
 LINE STEP(-30, 0)-STEP(0, 5)
NEXT
```

In questo programma la prima istruzione `LINE` traccia un segmento verticale dall'angolo inferiore sinistro a quello superiore sinistro dello schermo (per i segni, teniamo presente il verso di crescita delle coordinate x e y indicato in Figura 34). Il primo ciclo `FOR...NEXT` traccia i gradini di scala dall'angolo superiore sinistro alla metà destra dello schermo, e il secondo ciclo `FOR...NEXT` traccia i segmenti verticali non raccordati dalla metà destra dello schermo all'angolo inferiore sinistro. L'uscita grafica è indicata in Figura 35.

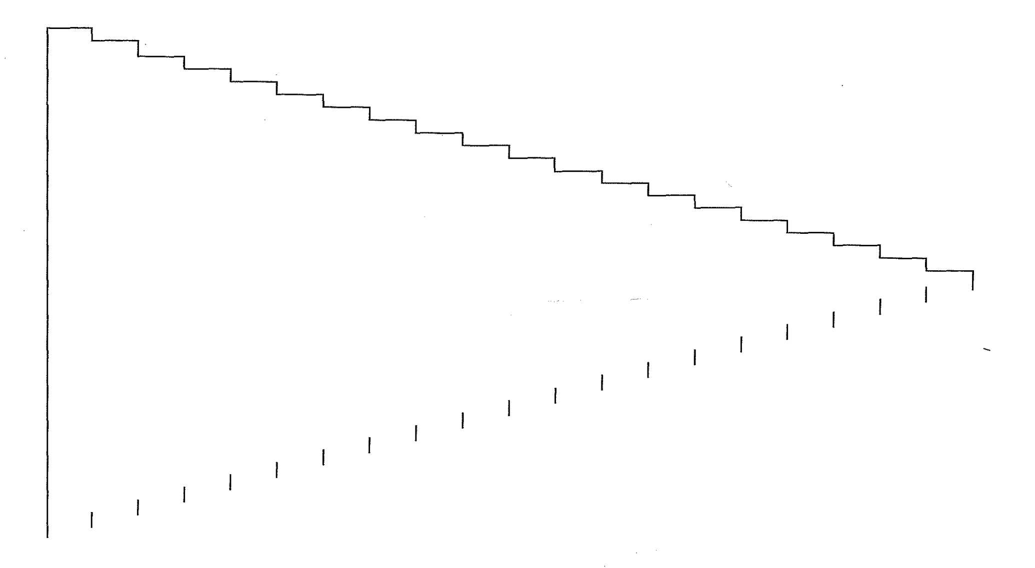


Figura 35 *- Effetto di un programma con istruzioni LINE*

DISEGNO DI RETTANGOLI (*B, BF*)

L'istruzione LINE nella forma vista finora permetterebbe di scrivere un semplice programma che collega quattro segmenti per formare un rettangolo, come il seguente BOX:

```
REM *** BOX ***
SCREEN 1
LINE (50, 50)-(170, 50)
LINE - STEP (0, 120)
LINE - STEP (-120, 0)
LINE - STEP (0, -120)
```

Tuttavia un modo più semplice per disegnare un rettangolo è fornito dall'opzione B (per Box = riquadro) dell'istruzione LINE, che richiede di indicare soltanto due vertici opposti (l'inferiore sinistro e il superiore destro, o viceversa) del rettangolo. L'esempio che segue produce lo stesso risultato del precedente programma BOX:

```
SCREEN 1
LINE (50, 50)-(170, 170), , B
```

Osserviamo che lo spazio fra le due virgole che precedono l'opzione B è riservato per un numero che determina il *colore* del perimetro del rettangolo, secondo quanto sarà indicato nel paragrafo "Colore di primo piano e di sfondo" (→ pag. 162).
Se alla fine dell'istruzione LINE si aggiunge l'opzione BF (per Fill = riempi), l'interno del rettangolo risulta colorato dello stesso colore del perimetro.

LINEE TRATTEGGIATE (*STILE*)

L'istruzione LINE può contenere un ulteriore parametro, detto *stile,* che permette di disegnare diversi tipi di linee tratteggiate o punteggiate. La forma sintattica è in questo caso:

```
LINE (x1, y1)-(x2, y2), , , stile
```

dove *stile* è un numero intero nel formato esadecimale (cioè preceduto dai caratteri "&H") che determina l'aspetto della linea, e si costruisce attraverso i passi seguenti:

1) si stabilisce quali pixel della linea devono essere disegnati e quali no, attribuendo ai primi il bit "1" e agli altri a bit "0" (la prova può essere eseguita su un foglio quadrettato)

2) si scrive una sequenza di 16 bit che rappresenta la linea desiderata; ad esempio, per la linea di figura 36, costituita alternativamente da due pixel disegnati e due no si avrebbe:

1100 1100 1100 1100

Figura 36 - *Linea punteggiata costituita alternativamente da due punti disegnati e due no*

3) si trasforma ciascun gruppo di 4 bit nel carattere esadecimale corrispondente secondo la Tabella 24; in questo caso si avrebbero i caratteri:

CCCC

che, preceduti dai caratteri "&H", costituiscono il numero esadecimale desiderato.

Gruppo di 4 bit	Carattere esadecimale	Gruppo di 4 bit	Carattere esadecimale
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Tabella 24 - *Corrispondenza fra gruppi di quattro bit e caratteri esadecimali*

Il programma LINEE che segue produce 6 tipi diversi di linee tratteggiate, e la Figura 37 mostra il suo effetto.

```
REM *** LINEE ***
SCREEN 2
DATA &HCCCC, &HFF00, &HF0F0
DATA &HF000, &H7777, &H8888
Riga% = 4
Colonna% = 4
Sx% = 60
Dx% = 600
Y% = 28
FOR I% = 1 TO 6
 READ Stile%
 LOCATE Riga%, Colonna%
 PRINT HEX$(Stile%)
 LINE (Sx%, Y%)-(Dx%, Y%), , , Stile%
 Riga% = Riga% + 3
 Y% = Y% + 24
NEXT
```

```
CCCC.................................................................
FFØØ- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
FØFØ-----------------------------------------------------------------
FØØØ- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
7777 ----------------------------------------------------------------
8888.................................................................


Premere un tasto per continuare
```

Figura 37 - Effetto del programma LINEE che genera linee tratteggiate

DISEGNO DI CERCHI ED ELLISSI (CIRCLE)

QBASIC permette di disegnare un gran numero di figure circolari, ellittiche od ovali, oltre ad archi e torte con fette staccate, tramite l'istruzione CIRCLE. Con varianti di questa istruzione si possono disegnare praticamente tutti i tipi di linee curve.
Come è intuitivo, per disegnare un cerchio è necessario conoscere la posizione del suo centro e la lunghezza del raggio, e queste informazioni sono richieste anche da QBASIC. Infatti, la più semplice forma sintattica dell'istruzione CIRCLE è

```
CIRCLE [STEP] (x, y), raggio
```

dove: *x, y* sono le coordinate del centro e *raggio* è la distanza dei suoi punti dal centro.
L'esempio che segue disegna un cerchio con centro (300, 100) e raggio 50.

```
SCREEN 2
CIRCLE (300, 100), 50
```

Osserviamo che se vogliamo usare come origine delle coordinate il centro dello schermo anziché l'angolo superiore sinistro, l'ultima istruzione va riscritta come

```
CIRCLE STEP (-20, 0), 50
```

DISEGNO DI ELLISSI (*ASPETTO*)

L'istruzione CIRCLE può contenere un ulteriore parametro, detto *aspetto,* che modifica il cerchio in un'ellisse. In questo caso la forma sintattica diventa

```
CIRCLE [STEP] (x, y), raggio, , , , aspetto
```

dove: le virgole tra *raggio* e *aspetto* delimitano altri parametri che vedremo tra poco, mentre *aspetto* è il rapporto tra l'asse verticale e l'asse orizzontale dell'ellisse:

$$\text{aspetto} = \frac{\text{asse verticale}}{\text{asse orizzontale}}$$

Se *aspetto* è maggiore di 1, l'ellisse risulta allungata in direzione dell'asse verticale, se è compreso tra 0 e 1 l'ellisse risulta appiattita sull'asse orizzontale, come mostrano i due esempi di Figura 38.

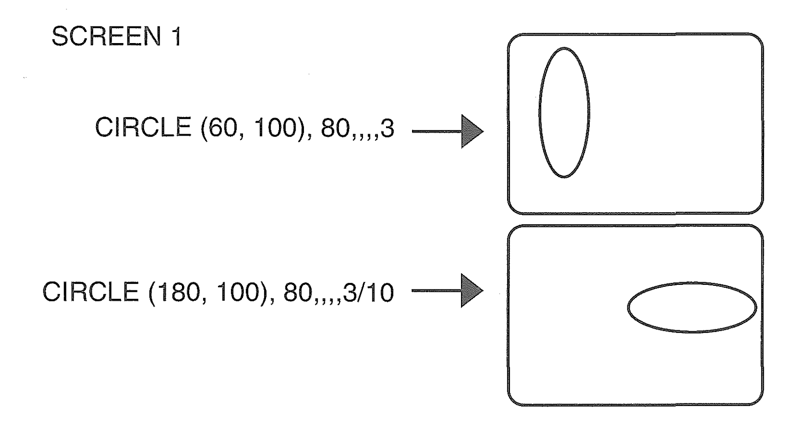


Figura 38 - Due diversi valori del parametro aspetto, *e i relativi effetti prodotti*

DISEGNO DI ARCHI

Un arco è un tratto di ellisse, i cui estremi vanno indicati specificando gli angoli da cui sono "visti" dal centro dell'ellisse.
L'istruzione CIRCLE (al pari delle funzioni trigonometriche COS, SIN, TAN), misura gli angoli non in gradi ma in radianti, che possono essere calcolati in base alla semplice formula

$$\text{radianti} = \text{gradi} \cdot \frac{\pi}{180}$$

(dove il valore di π, nel caso in cui debba essere impiegato più volte, può essere assegnato una volta per tutte a una costante numerica con un'istruzione del tipo

```
CONST PI = 3.141592653589#).
```

Si tenga anche presente che l'ampiezza di un angolo viene misurata a partire dal semiasse x positivo (corrispondente alla posizione delle ore 3 su un orologio) e in verso antiorario: la Figura 39 indica i valori di alcuni angoli notevoli.

La forma sintattica dell'istruzione CIRCLE per disegnare archi è allora

CIRCLE [STEP] (*x*, *y*), *raggio*, , *inizio*, *fine*[, *aspetto*]

Un esempio del suo funzionamento è fornito dal seguente programma

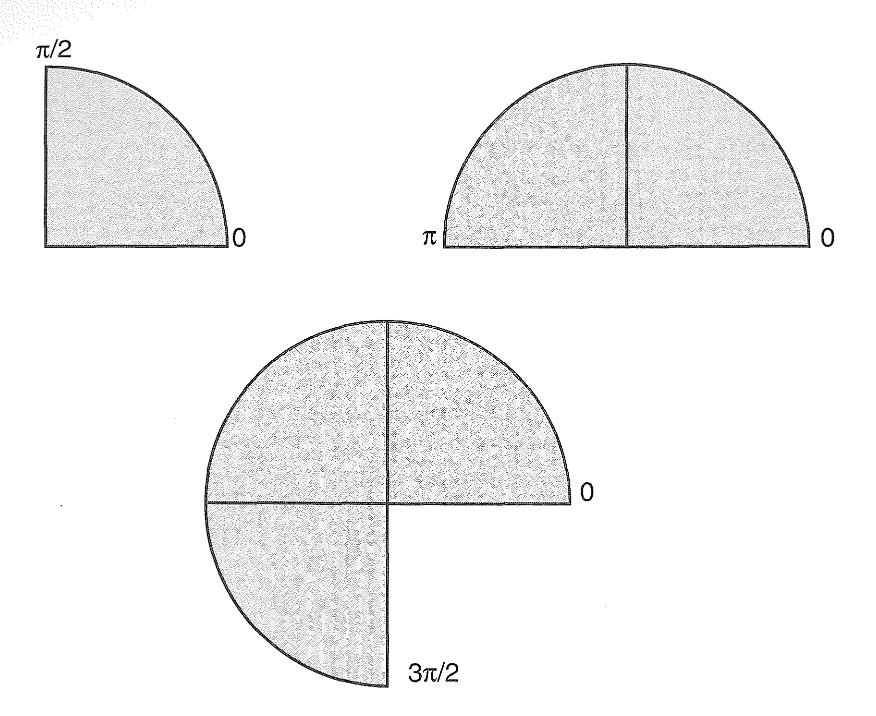


__Figura 39__ - Misura degli angoli nell'istruzione CIRCLE

```
SCREEN 2
CLS
CONST PI = 3.141592653589#
Inizio = 0
FOR Raggio% = 100 TO 220 STEP 20
 Fine = Inizio + (PI / 2.01)
 CIRCLE (320, 80), Raggio%, Inizio, Fine
 Inizio = Inizio + (PI / 4)
NEXT Raggio%
```

Esso disegna i sette archi indicati in Figura 40, a partire da quello più piccolo e interno che inizia "alle ore 3" e termina "alle ore 12" (in realtà per ottenere le Figure 40 e 41 ho aggiunto alla fine dell'istruzione CIRCLE il valore .5, per compensare la deformazione orizzontale prodotta dalla mia stampante).

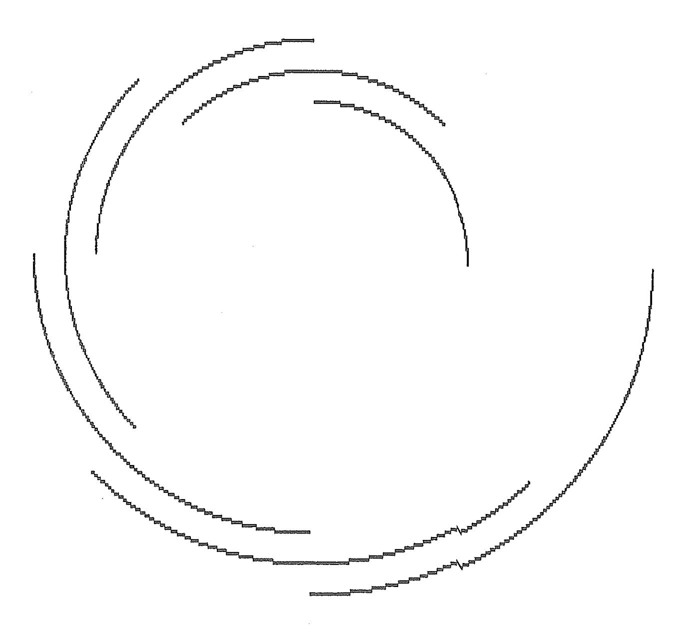


Figura 40 - Archi di cerchio ottenuti con l'istruzione CIRCLE

DISEGNO DI UNA TORTA

Se nell'istruzione CIRCLE uno degli argomenti *inizio* o *fine* è negativo, l'inizio o la fine dell'arco risulta unito con un raggio al centro. Se entrambi gli argomenti sono negativi, si possono disegnare forme che vanno da una fetta di torta alla torta da cui è stata tagliata la fetta. Il programma FETTA che segue disegna la torta con una fetta staccata mostrata in Figura 41.

```
REM *** FETTA ***
SCREEN 2
CONST Raggio = 150, PI = 3.141592653589#
Inizio = 2.5
Fine = PI
CIRCLE (320, 100), Raggio, , -Inizio, -Fine
CIRCLE STEP (70, 10), Raggio, , -Fine, -Inizio
```

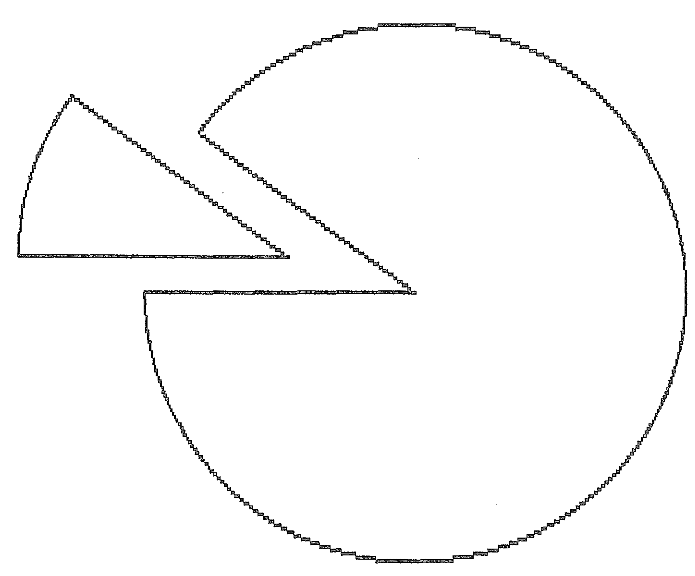


Figura 41 - Torta con fetta staccata ottenuta con l'istruzione CIRCLE

DEFINIZIONE DI UNA FINESTRA GRAFICA (CLS 1, VIEW)

Gli esempi grafici presentati finora hanno usato l'intero schermo video come tavola da disegno; tuttavia l'istruzione VIEW permette di definire all'interno dello schermo fisico una specie di schermo in miniatura, detto "finestra grafica". Una volta definita, tutte le operazioni grafiche avvengono all'interno di questa finestra, e i tentativi di disegnare al suo esterno sono ignorati.
L'uso di una finestra grafica presenta due vantaggi:

- rende facile cambiare la grandezza e la posizione dell'area di schermo in cui si vedono i grafici
- permette di cancellare lo schermo all'interno della finestra, senza coinvolgere il suo esterno, con l'istruzione

```
CLS 1
```

L'istruzione VIEW non consente l'opzione STEP, e ha la seguente forma sintattica:

```
VIEW [[SCREEN] (x1, y1) - (x2, y2)[,[colore],[bordo]]]
```

dove: (*x1, y1*) e (*x2, y2*) definiscono gli angoli della finestra, secondo la sintassi standard per i rettangoli di QBASIC vista nel paragrafo "Disegno di rettangoli" (→ pag. 153). Gli argomenti facoltativi *colore* e *bordo* permettono di scegliere un colore rispetti-

vamente per l'interno e il perimetro della finestra, come vedremo nel paragrafo "Colore di primo piano e di sfondo" (→ pag. 162).
In assenza dell'opzione SCREEN, tutte le coordinate di pixel sono considerate *relative* alla finestra anziché all'intero schermo. Ad esempio, se dopo l'istruzione

```
VIEW (50, 60) - (150, 175)
```

si disegna un pixel con l'istruzione

```
PSET (10, 10)
```

il pixel sarà visibile, in quanto è situato 10 pixel al di sotto e 10 pixel a destra dell'angolo superiore sinistro della finestra (le sue coordinate assolute di schermo sono (50+10, 60+10), cioè (60, 70).
Invece in presenza dell'opzione SCREEN tutte le coordinate sono considerate *assolute*, cioè misurate a partire dai lati dello schermo e non della finestra. Perciò, se dopo l'istruzione

```
VIEW SCREEN (50, 60) - (150, 175)
```

si disegna un pixel con l'istruzione

```
PSET (10, 10)
```

il pixel non sarà visibile, perché situato 10 pixel al di sotto e 10 pixel a destra dell'angolo superiore sinistro dello schermo, quindi al di fuori della finestra.
In assenza di tutti gli argomenti, l'istruzione VIEW considera come finestra l'intero schermo.

RIDEFINIZIONE DELLE COORDINATE DELLA FINESTRA (WINDOW)

Le coordinate considerate finora per posizionare i pixel sullo schermo rappresentano le effettive distanze fisiche dall'angolo superiore sinistro dello schermo (o della finestra), e sono dette perciò *coordinate fisiche*.
La loro origine, di coordinate (0, 0), è l'angolo superiore sinistro dello schermo (o della finestra), ed esse crescono spostandosi verso il basso e verso destra, come indicato nella Figura 34 del paragrafo risoluzione e pixel (→ pag. 147).
Per cambiare questo sistema di riferimento, e sostituirlo ad esempio con il più familiare N.B. piscel ≡ risoluzione ittica sistema di coordinate cartesiane, si usa l'istruzione

```
WINDOW [[SCREEN] (x1,y1) -(x2,y2)]
```

dove *y1, y2, x1* e *x2* sono numeri reali che indicano rispettivamente i lati alto, basso, sinistro e destro della finestra, e sono detti *coordinate logiche*. Ad esempio, le seguenti istruzioni

```
SCREEN 2
WINDOW (-320, 100)-(320, -100)
```

portano la posizione dell'origine del sistema di coordinate nel centro dello schermo, in modo che una successiva istruzione

```
CIRCLE (0, 0), 150
```

possa disegnare un cerchio con il centro nel centro dello schermo e raggio 150. Osserviamo che dopo l'istruzione WINDOW la coordinata verticale cresce verso l'*alto*, mentre dopo l'istruzione WINDOW SCREEN cresce verso il *basso* (in entrambi i casi la coordinata orizzontale cresce verso destra).
In assenza di tutti gli argomenti, WINDOW ripristina il sistema di coordinate fisiche standard.

USO DEL COLORE

Come ho accennato al paragrafo "Istruzione SCREEN" (→ pag. 148), se si dispone di un adattatore grafico CGA, si può selezionare una delle due seguenti modalità grafiche

- SCREEN 1 con risoluzione 320*200 pixel, 4 colori di primo piano e 16 di sfondo
- SCREEN 2 con risoluzione 640*200 pixel, un colore di primo piano e uno di sfondo.

Nei paragrafi successivi ci interesseremo in prevalenza della modalità SCREEN 1, dato che SCREEN 2 è una modalità monocromatica.

COLORE DI PRIMO PIANO E DI SFONDO (COLOR)

Le istruzioni grafiche viste finora, e cioè PSET, PRESET, LINE, CIRCLE, VIEW possono impiegare l'ulteriore parametro *colore*, che determina il colore di primo piano, secondo le seguenti forme sintattiche semplificate:

```
PSET (x, y), colore

PRESET (x, y), colore

LINE (x1, y1)-(x2, y2), colore[,B[F]]

CIRCLE (x, y), raggio, colore

VIEW (x1, y1)-(x2, y2), colore
```

L'effetto del parametro *colore* dipende dal numero della palette o tavolozza di colori selezionata in una precedente istruzione COLOR, che ha la seguente forma sintattica:

```
COLOR [sfondo][, palette]
```

e permette di selezionare anche il colore di sfondo.
Nella modalità SCREEN 1 il *colore* è un'espressione numerica che può assumere uno dei valori o "attributi" 0, 1, 2 o 3, e la *palette* il valore 0 o 1, con gli effetti indicati in Tabella 25.

Attributo di colore	con palette 0	con palette 1
0	colore sfondo	colore sfondo
1	verde	azzurro
2	rosso	magenta
3	marrone	bianco

Tabella 25 *- Colori risultanti dal valore del parametro* colore *e dal numero della palette nella modalità SCREEN 1*

Lo *sfondo* può assumere invece un valore compreso tra 0 e 15, con gli effetti indicati in Tabella 26.

Valore	colore
0	nero
1	blu
2	verde
3	ciano
4	rosso
5	magenta
6	marrone
7	bianco
8	grigio scuro
9	azzurro
10	verde chiaro
11	ciano chiaro
12	rosa
13	magenta chiaro
14	giallo chiaro
15	bianco brillante

Tabella 26 *- Colori di sfondo nella modalità SCREEN 1*

Il valore 0 determina lo stesso colore dello sfondo, e quindi un disegno invisibile; esso è utile per cancellare una figura senza dover cancellare l'intero schermo o la finestra grafica, come nel seguente programma ELLISSE1, che disegna un'ellisse e la cancella quando si preme un tasto:

```
REM *** ELLISSE1 ***
SCREEN 1
CIRCLE (100, 100), 80, 2, , , 3
Pausa$ = INPUT$(1)
CIRCLE (100, 100), 80, 0, , , 3
```

Questa tecnica è impiegata per ottenere effetti di animazione, come nel seguente programma ELLISSE2.

```
REM *** ELLISSE2 ***
SCREEN 1
FOR K% = 50 TO 250
 CIRCLE (K%, 60), 40, 2, , , .5
 CIRCLE (K%, 60), 40, 0, , , .5
NEXT K%
CIRCLE (K%, 60), 40, 2, , , .5
```

Esso disegna e cancella ripetutamente un'ellisse, spostandone ogni volta verso destra la coordinata orizzontale del centro.

CAMBIAMENTO DEL COLORE DI PRIMO PIANO O DI SFONDO (PALETTE E PALETTE USING)

Se si dispone di una scheda CGA, i colori di ognuna delle due *palette* sono predeterminati, secondo quanto indicato in Tabella 25. Se invece si dispone di una scheda EGA o VGA, è possibile scegliere il colore corrispondente a ognuno dei quattro attributi ammessi, per mezzo delle istruzioni PALETTE e PALETTE USING. Esse permettono di assegnare qualsiasi colore della palette disponibile a qualsiasi attributo.
Ad esempio, l'istruzione seguente

```
PALETTE 4, 13
```

fa sì che i risultati di tutte le istruzioni grafiche che usano l'attributo 4 siano visualizzati in magenta chiaro, (che è il colore 13).

Questo cambiamento di colore è istantaneo, e riguarda anche le visualizzazioni grafiche già presenti sullo schermo.
Il seguente programma QUADRATI mostra l'effetto delle istruzioni PALETTE e PALETTE USING, visualizzando due rettangoli che cambiano continuamente di colore, insieme allo sfondo.

```
REM ***QUADRATI***
PALETTE 0, 1
SCREEN 1
FOR i% = 0 TO 3
 a%(i%) = i%
NEXT i%
LINE (138, 35)-(288, 165), 3, BF
LINE (20, 10)-(160, 100), 2, BF
DO
 FOR i% = 0 TO 3
  a%(i%) = (a%(i%) + 1) MOD 16
 NEXT i%
 PALETTE USING a%(0)
 FOR k% = 1 TO 10000: NEXT k%
LOOP WHILE INKEY$ = ""
```

Osserviamo che le due istruzioni LINE terminano con l'opzione BF che, come abbiamo visto nel paragrafo "Disegno di rettangoli" (→ pag. 153), determina il tracciamento di un rettangolo riempito con lo stesso colore del perimetro.

RIEMPIMENTO DI FIGURE (PAINT)

Se si vogliono riempire figure di forma qualsiasi si usa l'istruzione PAINT, che ha la forma sintattica seguente:

`PAINT [STEP](`*x*`, `*y*`)[,[`*interno*`], [`*contorno*`]]`

dove: x, y sono le coordinate di un punto qualsiasi interno alla figura da riempire, *contorno* è il numero del colore del bordo, e *interno* ha un significato che varia leggermente se il riempimento deve avvenire con un colore uniforme, oppure con un motivo personalizzato dall'utente, come vedremo nelle prossime pagine.

RIEMPIMENTO CON UN COLORE

Per il riempimento con un colore, *interno* è il numero del colore con cui si vuole dipingere la figura. Ad esempio, il seguente programma disegna un cerchio in ciano, quindi ne dipinge l'interno in magenta:

```
SCREEN 1
CIRCLE (160, 100), 50, 1
PAINT (160, 100), 2, 1
```

(naturalmente le coordinate che compaiono nella precedente istruzione PAINT potrebbero essere sostituite da quelle di un qualsiasi altro punto interno, come ad esempio (150, 90), (170, 110) o (180, 80)).
È bene tenere presenti i seguenti punti:

- se le coordinate di PAINT indicano un punto *sul contorno* della figura, non si ottiene alcun riempimento, mentre se indicano un punto esterno alla figura, viene dipinto tutto lo schermo tranne l'interno della figura (che rimane con il colore dello sfondo corrente);
- la figura dev'essere completamente chiusa, altrimenti il colore "esce fuori" riempiendo l'intero schermo, oppure una figura più ampia che racchiude la precedente. Si può provare con il seguente programma, nel quale una circonferenza con una piccola interruzione è racchiusa da un rettangolo; il colore comincia a riempire l'interno della circonferenza, e continua a riempire l'intero rettangolo;

```
SCREEN 2
CONST PI = 3.141592653589#
CIRCLE (300, 100), 80, , 0, 1.9 * PI, 1/3
LINE (200, 10)-(400, 190), , B

PAINT (300, 100)
```

- se si dipinge un oggetto con un colore diverso dal bordo, si deve usare l'opzione *contorno,* come nel seguente esempio che disegna un triangolo in verde e ne dipinge l'interno in rosso.

```
REM *** TRIANGOLO ***
SCREEN 1
COLOR , 0
LINE (10, 25)-(310, 25), 1
LINE -(160, 175), 1
LINE -(10, 25), 1
PAINT (160, 100), 2, 1
```

(se invece si scrivesse PAINT (160, 100), 2 l'intero schermo verrebbe dipinto di rosso).

RIEMPIMENTO CON MOTIVO MONOCROMATICO

Per il riempimento con un motivo creato dall'utente, *interno* è un'espressione di stringa della seguente forma:

```
CHR$(num1) + CHR$(num2) + ... + CHR$(numn)
```

dove *num1, num2, ... numn* sono dei numeri nel formato esadecimale (cioè preceduti dai caratteri "&H"), che si determinano in maniera diversa a seconda che si voglia ottenere un motivo monocromatico (nella modalità SCREEN 2) oppure un motivo a colori (nella modalità SCREEN 1).
Per costruire un motivo monocromatico si seguono i passi seguenti:

1) si disegna il motivo di riempimento tracciando dei segni qualsiasi su un foglio di carta quadrettata avente 8 colonne e fino a 64 righe; un esempio può essere quello di Figura 42.

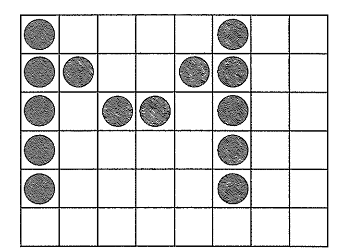


Figura 39 *- Schema di un motivo di riempimento monocromatico*

2) si trasforma ogni quadratino contenente un segno in un bit "1", e ogni quadratino vuoto in un bit "0", ottenendosi tanti gruppi di 8 bit per quante sono le righe di Figura 42. Il risultato è indicato in Figura 43

1	0	0	0	0	1	0	0
1	1	0	0	1	1	0	0
1	0	1	1	0	1	0	0
1	0	0	0	0	1	0	0
1	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0

Figura 43 *- Trasformazione in bit di un motivo di riempimento*

3) si trasforma ciascun gruppo di 4 bit nel carattere esadecimale corrispondente, secondo quanto indica la Tabella 24 di pag. 154, ottenendosi quindi:

```
1000 0100 →  &H84
1100 1100 →  &HCC
1011 0100 →  &HB4
1000 0100 →  &H84
1000 0100 →  &H84
0000 0000 →  &H00
```

4) Si utilizzano i numeri esadecimali così determinati come argomenti di funzioni CHR$, che vengono concatenate per costruire un'espressione di stringa del tipo

```
Motivo$ = CHR$(&H84) + CHR$(&HCC) + CHR$(&HB4) +
CHR$(&H84) + CHR$(&H84) + CHR$(&H0)
```

Questa espressione può finalmente essere usata come valore di *interno* in un'istruzione

```
PAINT STEP(0, 0), Motivo$
```

Un programma completo può essere il seguente CERCHIOEMME:

```
REM ***CERCHIOEMME***
SCREEN 2
CLS
Motivo$ = CHR$(&H84) + CHR$(&HCC) + CHR$(&HB4) +
CHR$(&H84) + CHR$(&H84) + CHR$(&H0)
CIRCLE STEP(0, 0), 150
PAINT STEP(0, 0), Motivo$
```

che produce l'uscita di Figura 44.

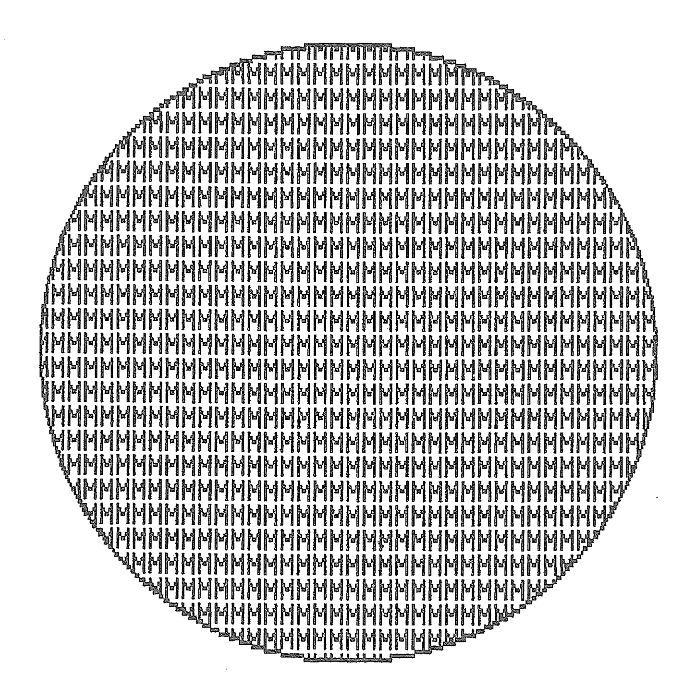


Figura 44 *- Un cerchio riempito con un motivo definito dall'utente*

RIEMPIMENTO CON UN MOTIVO A COLORI

Nella modalità SCREEN 1 è possibile creare un motivo colorato costituito da strisce diagonali di colore verde e rosso (con la palette 0), oppure azzurro e magenta (con la palette 1).
I passi da seguire sono i seguenti:

1) si disegna il motivo su un foglio di carta quadrettata avente 4 colonne e fino a 64 righe (ciò perché nella modalità SCREEN 1 ogni pixel richiede 2 bit, mentre una riga di pixel è sempre memorizzata in un numero intero di 8 bit). Un esempio può essere quello di Figura 45

azzurro	magenta	magenta	magenta
magenta	azzurro	magenta	magenta
magenta	magenta	azzurro	magenta
magenta	magenta	magenta	azzurro

Figura 45 *- Schema di un motivo di riempimento colorato*

2) si trasforma il colore di ciascun quadratino nel suo codice binario, che si ottiene convertendo in numero binario l'attributo di colore indicato in Tabella 25 (0 → 00; 1 → 01; 2 → 10; 3 → 11). Nel nostro caso si hanno i valori indicati in Figura 46

01	10	10	10
10	01	10	10
10	10	01	10
10	10	10	01

Figura 46 *- Trasformazione in bit di un motivo di riempimento*

3) si trasforma ciascun gruppo di 4 bit nel carattere esadecimale corrispondente, secondo la solita Tabella 24 di pag. 154, ottenendo si quindi:

```
0110 1010 → &H6A
1001 1010 → &H9A
1010 0110 → &HA6
1010 1001 → &HA9
```

4) si utilizzano i numeri esadecimali così determinati come argomenti di funzioni CHR$, che vengono concatenate per costruire un'espressione di stringa, da usare come valore di *interno* in un'istruzione PAINT.

Il seguente programma disegna un triangolo e ne dipinge l'interno con il motivo creato in precedenza

```
SCREEN 1
Motivo$ = CHR$(&H6A) + CHR$(&H9A) + CHR$(&HA6) +
CHR$(&HA9)
LINE (10, 25)-(310, 25)
LINE -(160, 175)
LINE -(10, 25)
PAINT (160, 100), Motivo$
```

Osserviamo che se si vuole dipingere una figura bordata con un colore contenuto anche nel motivo, nell'istruzione PAINT si deve indicare anche l'argomento del *contorno,* altrimenti il motivo fuoriesce dai confini della figura. Perciò nel programma seguente l'istruzione PAINT contiene anche l'argomento 2.

```
SCREEN 1
Motivo$ = CHR$(&H6A) + CHR$(&H9A) + CHR$(&HA6) +
CHR$(&HA9)
LINE (10, 25)-(310, 25), 2
LINE -(160, 175), 2
LINE -(10, 25), 2
PAINT (160, 100), Motivo$, 2
```

LINGUAGGIO DI MACRO GRAFICHE (DRAW)

La più potente istruzione grafica di QBASIC è DRAW, che ha la seguente forma sintattica:

```
DRAW stringacomandi
```

dove la *stringacomandi* contiene uno o più comandi che permettono di disegnare grafici e figure sullo schermo. I *comandi* o *macro* sono formati da una o due lettere, e costituiscono nel loro insieme un linguaggio per la definizione di oggetti grafici detto **linguaggio di macro grafiche** o GML (da Graphics Macro Language).
La Tabella 27 elenca i comandi che permettono di tracciare righe e muovere il cursore (se n% non è indicato, il cursore si muove di un pixel), e la Figura 47 riassume graficamente i loro effetti.

Comando	Effetto
B	Muove il cursore senza disegnare
N	Disegna e riporta il cursore alla posizione di partenza
U [*n%*]	Muove il cursore in alto di *n%* pixel
D [*n%*]	Muove il cursore in basso di *n%* pixel
L [*n%*]	Muove il cursore a sinistra di *n%* pixel
R [*n%*]	Muove il cursore a destra di *n%* pixel
E [*n%*]	Muove il cursore in diagonale a destra in alto di *n%* pixel
F [*n%*]	Muove il cursore in diagonale a destra in basso di *n%* pixel
G [*n%*]	Muove il cursore in diagonale a sinistra in basso di *n%* pixel
H [*n%*]	Muove il cursore in diagonale a sinistra in alto di *n%* pixel
M*x%*,*y%*	Muove il cursore al punto di coordinate assolute *x%*,*y%*
M + *x*,*y*M - *x*,*y*	Muove il cursore al punto di coordinate *x%*,*y%* relative alla posizione del pixel corrente

Tabella 27 *- Comandi di DRAW per tracciare righe e muovere il cursore*

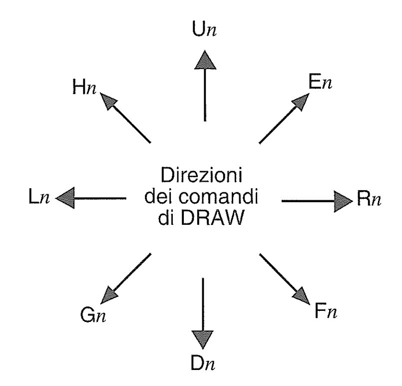


Figura 47 *- Direzioni dei comandi di DRAW*

Il programma DISLINEE che segue permette di disegnare linee verticali e orizzontali usando i tasti di movimento del cursore; esso contiene anche la macro B, che viene eseguita quando si preme la barra spaziatrice, e attiva/disattiva la visualizzazione del disegno, impostando la lunghezza del movimento unitario del cursore.

```
REM *** DISLINEE ***
CONST Su = 72, Giu = 80, Sx = 75, Dx = 77
CONST Spazio = " "
Z$ = CHR$ (0)
P$ = " "
PRINT "Usare i tasti cursore per disegnare linee"
PRINT " Premere la barra spaziatrice per iniziare/finire
il disegno di linee"
PRINT "Premere <Invio> per iniziare, <f> per finire"
DO
 LOOP WHILE INKEY$ = " "
 SCREEN 1
 DO
  SELECT CASE Risp $
   CASE Z$ + CHR$ (Su)
    DRAW P$ + C"C1 U2"
   CASE Z$ + CHR$ (Giu)
    DRAW P$ + "C1 D2"
   CASE Z$ + CHR$ (Sx)
    DRAW P$ + "C2 L2"
   CASE Z$ + CHR$ (Dx)
    DRAW P$ + "C2 R2"
   CASE Spazio
    IF P$ = " " THEN P$ = "B" ELSE P$ = " "
   CASE ELSE
  END SELECT
  Risp$ = INKEY$
 LOOP UNTIL Risp$ = "f"
 SCREEN 0, 0
 WIDTH 80
 END
```

Altri comandi, relativi a colore, rotazione e riduzione di scala, sono indicati in Tabella 28. Tra essi è molto utile in particolare il comando X, che permette di far eseguire a DRAW una *stringacomandi* definita in precedenza, assegnandola come argomento alla variabile VARPTR$ secondo la seguente forma sintattica:

```
DRAW "X" + VARPTR$ (stringacomandi)
```

Un esempio di utilizzo del comando X è fornito dal seguente programma TRIANGOLO:

```
REM *** TRIANGOLO ***
SCREEN 1
Triangolo$ = "F60 L120 E 60"
DRAW "C2 X" + VARPTR$ (Triangolo $)
DRAW "BD30 P1,2 C3 BM-30,-30"
```

Comando	Effetto
An%	Ruota un oggetto di *n%* * 90 gradi, dove n% può essere uguale a 0, 1, 2 o 3
Cn%	Imposta il colore della riga tracciata, dove n% è un attributo di colore
Pn1%,n2%	Imposta il colore di riempimento con l'attributo *n1%* e il colore del bordo di un oggetto con l'attributo *n2%*
Sn%	Determina la scala di grandezza del disegno, impostando la lunghezza del movimento unitario del cursore. Il valore di n% preimpostato è uguale a 4, equivalente a 1 pixel
TAn%	Ruota un angolo di n% gradi, da -360 fino a 360
X	Permette di eseguire una stringa di comandi definita in precedenza, assegnandola come argomento alla variabile VARPTR$

Tabella 28 *- Comandi di DRAW per colore, rotazione e riduzione di scala*

STAMPA DI GRAFICI (GRAPHICS)

Un'altra differenza tra le modalità testo e grafica riguarda il modo di stampare i risultati delle elaborazioni: mentre il testo può essere inviato direttamente alla stampante anziché al video con l'istruzione LPRINT (vista al paragrafo "Stampa su carta", → pag. 32), la grafica deve prima essere visualizzata sullo schermo, e poi trasferita alla stampante con la pressione dei tasti <Shift>+<PrtSc>. Perché questa operazione abbia successo, è però necessario che in precedenza sia stato eseguito il seguente comando di MS-DOS

`C:\DOS\GRAPHICS` *`tipo`*

dove *tipo* specifica il modello di stampante in uso, secondo quanto indicato in Tabella 29 (se si usa una stampante non compresa nell'elenco, si deve cercare nel suo Manuale a quale di quelle indicate essa corrisponde).

Nota. Se non si vuole "sporcare" l'immagine grafica con la scritta `"Premere un tasto per continuare"` presentata da QBASIC (ad esempio perché si vuole stamparla), è possibile inserire alla fine del programma grafico la riga

```
DO: LOOP WHILE INKEY$ = ""
```

che sospende l'elaborazione (e quindi la visualizzazione della scritta) finché non si preme un tasto qualsiasi.

Tipo	Stampante
COLOR 1	IBM Personal Computer Color Printer con nastro nero
COLOR4	IBM Personal Computer Color Printer con nastro rosso, verde, blu e nero
COLOR8	IBM Personal Computer Color Printer con nastro ciano, magenta, giallo e nero
HPDEFAULT	Qualsiasi stampante Hewlett-Packard PCL
DESKJET	Hewlett-Packard DeskJet
GRAPHICS	IBM Personal Graphics Printer, Proprinter o Quietwriter
GRAPHICSWIDE	IBM Personal Graphics Printer con carrello da 11 pollici
LASERJET	Hewlett-Packard LaserJet
LASERJETII	Hewlett-Packard LaserJet II
PAINTJET	Hewlett-Packard Paint Jet
QUIETJET	Hewlett-Packard QuietJet
QUIETJETPLUS	Hewlett-Packard QuierJet Plus
RUGGEDWRITER	Hewlett-Packard RuggedWriter
RUGGERWRITERWIDE	Hewlett-Packard RuggedWriterwide
THERMAL	IBMPC Convertible Thermal Printer
THINKJET	Hewlett-Packard ThinkJet

Tabella 29 *- Tipi di stampante che si possono indicare nel comando GRAPHICS di MS-DOS*

Capitolo 15

Il suono



ISTRUZIONE BEEP

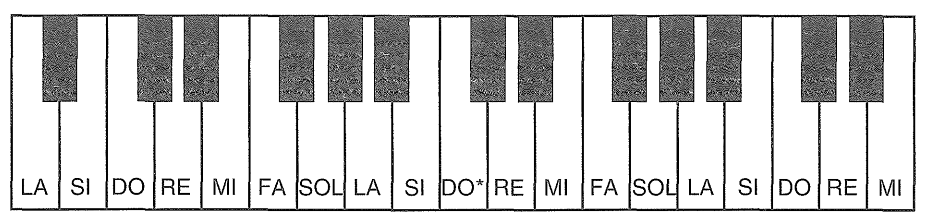
QBASIC permette di generare semplici motivi musicali che, a causa delle caratteristiche hardware dei primi personal computer (e della maggior parte di quelli attuali), sono limitati a una sola nota alla volta. SOUND (che produce risultati piuttosto semplici) e PLAY (che si articola in un vero e proprio linguaggio musicale).
A parte l'istruzione BEEP (o PRINT CHR$ (7)), che determina l'emissione di un suono di frequenza 800 Hertz (o cicli al secondo) e durata un quarto di secondo, sono disponibili le istruzioni musicali SOUND (che produce risultati piuttosto semplici) e PLAY (che si articola in un vero e proprio linguaggio musicale).

ISTRUZIONE SOUND

L'istruzione SOUND produce una nota simile a quella di un singolo tasto del pianoforte. La sua forma sintattica è:

```
SOUND frequenza, durata
```

dove: *frequenza* è un numero compreso fra 37 e 32767, che rappresenta il numero di Hertz della nota, secondo quanto indicato in Figura 48; *durata* è un numero intero compreso fra 0 e 65535 che rappresenta il numero di impulsi di clock della nota desiderata (un secondo equivale a 18,2 impulsi).



246.94 293.66 349.23 440.00 523.25 659.26 783.99 987.77 1174.70
220.00 261.63 329.63 392.00 493.88 587.33 698.46 880.00 1046.50 1318.50

Figura 48 *- Valori della frequenza per le note centrali del pianoforte (DO* è il do centrale)*

La *frequenza* delle altre note si ottiene considerando che essa raddoppia nel passare da un'ottava alla successiva. Per quanto riguarda la *durata*, la Tabella 30 mostra i suoi valori in relazione ai termini musicali "tempo" e "notazione".

Tempo	Notazione	Durata
Molto lento	Largo Adagio Larghetto	27.3 - 18.02 18.2 - 18.55 16.55 - 14.37
Lento	Andante	14.37 - 10.11
Medio	Moderato	10.11 - 9.1
Veloce	Allegro	9.1 - 6.5

Tabella 30 - *Corrispondenza fra la terminologia musicale e i valori del parametro* durata

L'istruzione SOUND è usata soprattutto per produrre suoni glissati o simili a quelli di una sirena, inserendola in cicli del tipo

```
FOR n = 37 TO 32767
 SOUND n, 1
NEXT
```

In pratica può essere opportuno aggiungere all'istruzione FOR un'opzione del tipo STEP 5, per rendere distinguibili le singole note, ridurre la durata del suono a valori inferiori a 1, e restringere il campo di variabilità della frequenza, che a valori abbastanza elevati produce suoni non udibili.
L'uso della funzione RND permette di ottenere suoni di "tipo computer", come quelli impiegati in alcuni videogiochi: in questo caso il programma può essere simile al seguente SUONOGAME, che produce note di frequenza e durata casuale.

```
REM ***SUONOGAME***
FOR Tempo = 1 TO 50
 Nota = INT(RND * 2735) + 500
 Dur = INT(RND * 3) + 1
 SOUND Nota, Dur
NEXT Tempo
```

LINGUAGGIO DI PROGRAMMAZIONE MUSICALE PLAY

L'istruzione PLAY è molto più versatile di SOUND per la creazione di semplici motivi musicali e mette a disposizione un piccolo linguaggio di programmazione musicale. Questo permette di costruire una opportuna *stringacomandi* da inserire nell'istruzione

```
PLAY stringacomandi
```

Il linguaggio comprende comandi di ottava, di durata, di tempo e di operazioni, che sono elencati qui di seguito.

Comandi di ottava

>
Suona la nota che segue all'ottava immediatamente più alta. Il massimo valore che può raggiungere un'ottava è 6.

<
Suona la nota che segue all'ottava immediatamente più bassa. Il minimo valore che può raggiungere un'ottava è 0.

O n
Imposta come ottava corrente quella numero n, (dove n, può variare da 0 a 6). Il valore predefinito è n = 4. Il do centrale (DO*) è all'inizio dell'ottava numero 3.

A, B, C, D, E, F, G, [#+-]
Sono i nomi delle note nella notazione anglossassone, e determinano il suono della rispettiva nota.
La Tabella 31 riporta la corrispondenza con i nomi delle note nella notazione dei Paesi latini.

Paesi latini	**Paesi Anglosassoni**
LA	A
SI	B
DO	C
RE	D
MI	E
FA	F
SOL	G

Tabella 31 *- Corrispondenza fra i nomi delle note nei Paesi latini e anglosassoni*

L'aggiunta a una nota del simbolo "#" o "+" produce un diesis (mezzo tono in alto), l'aggiunta di "–" produce un bemolle (mezzo tono in basso).

N *n*
Suona la nota numero *n,* con *n* compreso fra 0 e 84. *n* = 0 indica una pausa.

Comandi di durata

L*n*
Imposta la durata di ogni nota, come indicato in Tabella 32.

comando	durata	denominazione	figura di valore
L1	4/4	semibreve	
L2	3/4	semibreve con “	
L3	2/4	minima	
L4	1/4	semiminima	
L8	1/8	croma	
L16	1/16	semicroma	
L32	1/32	bicroma	
L64	1/64	semicroma	

Tabella 32 *- I comandi di durata e le figure di valore prodotte*

Quando la durata precede una nota (come ad esempio in LC2), questa durata ha effetto su tutte le note che seguono fino al prossimo comando L. Se si fa seguire a una nota un valore *n* senza indicare la lettera L, (scrivendo ad esempio C2), solo quella nota è impostata a quella durata, mentre quelle che seguono mantengono la durata impostata in precedenza. Ad esempio, la stringa

```
"L4 C2 E G"
```

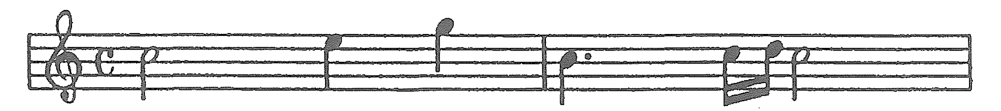
ha il seguente effetto: il comando L4 imposta la durata delle note successive a 1/4, mentre C2 imposta la durata della sola nota C a 2/4 (gli spazi sono del tutto irrilevanti, e servono solo per migliorare la leggibilità).
Il punto (.) posto dopo una nota (o una pausa) ne prolunga la durata della metà (cioè determina una nota o una pausa che ha una volta e mezza la durata determinata da L*n*); dopo una nota si può indicare più di un punto per aumentare la durata ancora di più.

MN
Attiva il modo *musicale normale,* nel quale ogni nota è suonata per sette ottavi del tempo determinato da L*n*.

ML
Attiva il modo *musicale legato,* nel quale ogni nota è suonata per l'intero tempo determinato da L*n*.

MS
Attiva il modo *musicale staccato,* nel quale ogni nota è suonata per tre quarti del tempo determinato da L*n*.

Esempio. Per trasformare in una *stringacomandi* il seguente brano musicale



è sufficiente impostare la durata L4, quindi scrivere le singole note, indicando le durate diverse da L4.
Osserviamo che la quarta nota (il si) appartiene all'ottava inferiore a quella di partenza, mentre le tre successive appartengono all'ottava superiore a quella del si appena suonato. L'istruzione completa è pertanto:

```
PLAY "L4 C2 E G <B.> L16 C D L2 C"
```

Comandi di tempo

P*n*
Determina una pausa, di durata variabile da 4/4 ($n = 1$) a 1/64 ($n = 64$). Le possibilità sono le stesse del comando L*n*, e in particolare si può usare il punto (.) per aumentare la durata della pausa.

T*n*
Imposta il tempo, cioè il numero *n* di note da un quarto (L4) che vengono suonate in un minuto. È una misura di tipo "metronomo", che dà il numero di battiti al secondo, con *n* che varia da 32 a 255, e ha per valore predefinito 255.

Comandi di operazione

MF
Forza la *stringacomandi* di PLAY a suonare in primo piano, in modo che nessun'altra istruzione di QBASIC possa essere eseguita fino al termine della musica.

MB
Forza la *stringacomandi* di PLAY a suonare in sottofondo, in modo che successive istruzioni di QBASIC possano essere eseguite finché la musica termina.

X

Permette di eseguire una *stringacomandi* definita in precedenza, assegnandola come argomento alla variabile VARPTR$ secondo la seguente forma sintattica:

```
PLAY "X" + VARPTR$(stringacomandi)
```

Un esempio di utilizzo del comando X è fornito dal seguente programma scale, che suona la scala musicale nelle 7 ottave disponibili.

```
REM ***SCALE***
Scala$ = "CDEFGAB"
PLAY "L16"
FOR I% = 0 TO 6
 PLAY "O" + STR$(I%)
 PLAY "X" + VARPTR$(Scala$)
NEXT I%
```

In esso la durata delle note è impostata a 1/16, e l'istruzione `PLAY "O" + STR$(I%)` inserita nel ciclo FOR...NEXT ha l'effetto di aumentare di 1 il valore dell'ottava corrente a ogni esecuzione del ciclo.

Capitolo 16

Intercettazione degli errori



PREVEDERE GLI ERRORI

Durante l'esecuzione di un programma l'utente può commettere un errore, ad esempio dimenticare di accendere la stampante in un programma di stampa, o chiedere di aprire un file in un sottoindirizzario inesistente. Ciò avrebbe l'effetto di visualizzare un messaggio sullo schermo e fermare l'applicazione in corso. Poiché l'utente normalmente non saprebbe porre rimedio a questa situazione, essa deve essere prevista dal programmatore, indicando la serie di azioni che si devono svolgere al suo verificarsi. Ciò si ottiene con la tecnica dell'**intercettazione degli errori,** che consiste nell'attivare l'intercettazione dell'errore, scrivere un'opportuna "routine di gestione degli errori", che svolge le azioni necessarie per correggere l'errore (ad esempio la visualizzazione di un messaggio sullo schermo), e quindi restituire il controllo al programma principale.

ATTIVAZIONE DELL'INTERCETTAZIONE DEGLI ERRORI (ON ERROR GOTO)

L'intercettazione degli errori è attivata all'interno di un modulo dall'istruzione

```
ON ERROR GOTO riga
```

dove *riga* è il numero o l'etichetta della prima riga della routine di gestione degli errori. Quando QBASIC incontra la precedente istruzione, ogni successivo errore di esecuzione all'interno di quel modulo determina un salto alla riga indicata.

Nota. Il numero di *riga* non dev'essere 0, se si vuole attivare una routine di gestione degli errori. Infatti l'istruzione `ON ERROR GOTO 0` ha due effetti diversi a seconda di dove si trova. Se si trova

- *al di fuori* di una routine di gestione degli errori, disabilita l'intercettazione degli errori; perciò ogni errore successivo fa terminare il programma, anziché eseguire la routine
- *all'interno* di una routine di gestione degli errori, (come nell'esempio successivo) fa visualizzare il messaggio standard di QBASIC e ferma il programma.

CODICI DI ERRORE (ERR)

QBASIC dispone della funzione

```
ERR
```

che fornisce un codice numerico corrispondente al tipo di errore che si è verificato durante l'esecuzione di un programma. Nello scrivere la routine di gestione degli errori si dovrà indicare un'azione appropriata per ciascun codice di errore che si prevede si possa verificare. La Tabella 33 riporta tutti i codici possibili e i corrispondenti significati (come si può osservare, solo alcuni di questi errori possono essere commessi dall'utente; gli altri, essendo errori di programmazione, non interessano nel contesto della gestione degli errori).

Cod.	Significato	Cod.	Significato
1	NEXT senza FOR	37	Numero degli argomenti non corrispondente
2	Errore di sintassi	38	Matrice non definita
3	RETURN senza GOSUB	20	È necessaria una variabile
4	Valori dell'istruzione DATA esauriti	50	Superamento limiti dell'istruzione FIELD
5	Chiamata di funzione non valida	51	Errore interno
6	Superamento limiti di calcolo (overflow)	52	Nome o numero del file errato
7	Memoria esaurita	53	File non trovato
8	Etichetta non definita	54	Modalità di accesso al file errata
9	Indice inferiore fuori limite	55	File già aperto
10	Definizione doppia	56	Istruzione FIELD attiva
11	Divisione per zero	57	Errore di ingresso/uscita su periferica
12	Non ammesso in modalità diretta	58	Il file esiste già
13	Tipo di dati non corrispondente	59	Lunghezza del record errata
14	Spazio di stringa esaurito	61	Disco pieno
16	Formula a stringa troppo complessa	62	Immissione dati oltre la fine del file
17	Impossibile continuare	63	Numero del record errato
18	Funzione non definita	64	Nome del file errato
19	Manca RESUME	67	Troppi file
20	RESUME senza errore	68	Periferica non disponibile
24	Fuori tempo massimo di periferica	69	Superamento limiti nel buffer comunicazioni
25	Errore di periferica	70	Permesso negato
26	FOR senza NEXT	71	Disco non pronto
27	La stampante ha esaurito la carta	72	Errore di supporto del disco
29	WHILE senza WEND	73	Caratteristica avanzata non disponibile
30	WEND senza WHILE	74	Tentativo di rinominare su altro disco
33	Etichetta doppia	75	Errore di accesso al percorso/file
35	Sottoprogramma non definito	76	Percorso non trovato

Tabella 33 *- Codici di errore e relativi significati*

ROUTINE DI GESTIONE DEGLI ERRORI

Una routine di gestione degli errori consiste delle tre parti seguenti:

1. il numero o l'etichetta di riga indicata nell'istruzione ON ERROR GOTO *riga,* che è la prima istruzione che il programma esegue dopo il verificarsi di un errore;
2. il corpo della routine, che identifica l'errore che ha causato il salto e svolge l'azione appropriata per ogni errore previsto;
3. una o più istruzioni RESUME o RESUME NEXT, che riportano il controllo al programma principale.

Una routine di gestione degli errori non si può trovare in una procedura SUB o FUNCTION, o in una funzione DEF FN. Inoltre, dev'essere situata in modo che il normale flusso dell'esecuzione non l'attraversi, ad esempio essendo preceduta da un'istruzione END.

Esempio il seguente programma STAMPADATI apre la stampante come se fosse un file (come abbiamo visto nel paragrafo "File sequenziali: creazione", → a pag. 130), e le invia i dati letti da una coppia di istruzioni READ...DATA. Il programma rimanda poi a una routine Gesterr se si verifica uno dei seguenti errori:

- sono stati esauriti i valori presenti nell'istruzione DATA (codice di errore numero 4)
- la stampante non risponde perché non è stata accesa (codice di errore numero 25)
- la stampante ha esaurito la carta (codice di errore numero 27)

In ciascuno di questi tre casi viene visualizzato un messaggio sullo schermo e restituito il controllo al programma principale, secondo due meccanismi differenti che vedremo in dettaglio nel paragrafo successivo.

```
REM ***STAMPADATI***
DATA Roma, New York, Tokyo, Londra, Mosca
CONST Falso = 0, Vero = NOT Falso, Finedati = Falso
ON ERROR GOTO Gesterr
OPEN "LPT1:" FOR OUTPUT AS #1
DO
 READ Mem$
 IF NOT Finedati THEN
  PRINT #1, Mem$
 ELSE
  EXIT DO
 END IF
LOOP
CLOSE #1
END
Gesterr:
SELECT CASE ERR
 CASE 4
  Finedati = Vero
  RESUME NEXT
 CASE 25
  PRINT "Accendi la stampante, e ";
  PRINT "premi un tasto per continuare"
  Pausa$ = INPUT$(1)
  RESUME
 CASE 27
  PRINT "La stampante ha esaurito la carta."
  PRINT "Aggiungi carta e premi un tasto"
  Pausa$ = INPUT$(1)
  RESTORE
  RESUME
 CASE ELSE
 ON ERROR GOTO 0
END SELECT
```

USCITA DALLA ROUTINE DI GESTIONE DEGLI ERRORI (RESUME NEXT)

Come abbiamo visto nell'esempio precedente, l'uscita dalla routine di gestione degli errori può avvenire con una delle due istruzioni `RESUME` o `RESUME NEXT`. Esse presentano le seguenti differenze.
`RESUME` fa ritornare alla stessa istruzione del programma che ha causato l'errore (vedi Figura 49).
Nella routine Gesterr è stata usata `RESUME` per ritornare all'istruzione `PRINT` che ha tentato di inviare i risultati alla stampante; ciò per fornire un'altra opportunità di stampare il valore in Mem$, dopo che la stampante presumibilmente è stata accesa.

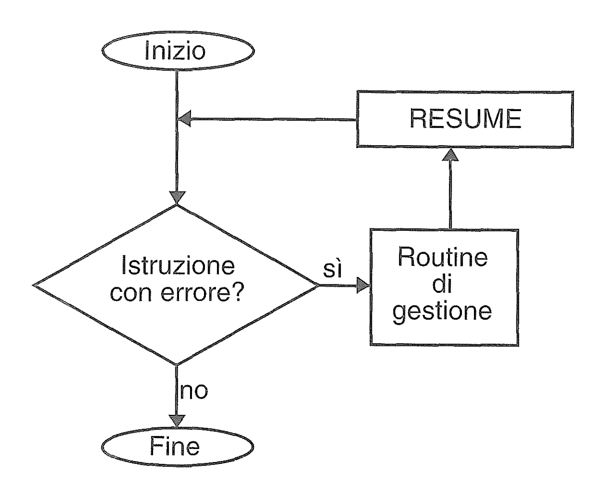


Figura 49 - Diagramma di flusso con l'istruzione RESUME

`RESUME NEXT` fa invece ritornare all'istruzione del programma successiva a quella che ha causato l'errore (vedi Figura 50).
Nella routine Gesterr è stata usata `RESUME NEXT` per ripristinare la situazione dall'errore. In questo caso sarebbe stato errato usare `RESUME`, perché essa avrebbe posto il programma in un ciclo senza fine, dato che ogni volta che il controllo ritorna all'istruzione `READ` del programma principale, si verifica un altro errore "Valori dell'istruzione DATA esauriti" che richiama di nuovo la routine, e così via.



Figura 50 - Diagramma di flusso con l'istruzione RESUME NEXT

Un'altra variante di RESUME ha la forma sintattica

```
RESUME riga
```

dove *riga* è il numero o l'etichetta di una riga esterna a qualsiasi blocco SUB...END SUB, FUNCTION...END FUNCTION o DEF FN...END DEF. A causa di queste restrizioni sulla posizione di *riga,* un'istruzione RESUME *riga* può dar luogo a effetti collaterali indesiderati se si trova in una routine di gestione degli errori che gestisce errori interni a una procedura SUB O FUNCTION, o a una funzione DEF FN. Per questa ragione è preferibile usare `RESUME` o `RESUME NEXT`.



LE COLLANE LIBRARIE DI TECNICHE NUOVE

COLLANA INFORMATICA

Autore/Titolo	Prezzo	ISBN 887081
Informatica generale		
Albanesi R., Boatti F., Bono G.R. - COGE - L'automazione della contabilità	L. 18.000	616 8
Andreini F. - Il controllo statistico di qualità con il PC	L. 29.000	550 1
Bartel R. - Norton utilities	L. 19.000	643 5
Benasi A.L. - Introduzione al personal computer	L. 11.000	161 1
Borrell J. - Quick Time Manuale d'uso	L. 32.000	0035 X
Computer e musica	L. 21.000	436 X
Fine L.H. - Sicurezza dei sistemi informativi	L. 25.000	466 1
Foster G. - SPSS - Guida iniziale all'analisi dei dati	L. 29.000	909 4
Gallippi A. - Dizionario di informatica inglese/italiano - italiano/inglese	L. 39.000	947 7
Gallippi A. - Introduzione all'informatica (4a ediz.)	L. 32.000	946 9
Gookin D., Townsend A. - Gestione dei dischi rigidi	L. 48.000	513 7
Gross M. - Norton Utilities 6 - In venti lezioni	L. 21.000	783 0
Gross M., Clark D. - Norton desktop per windows In venti lezioni	L. 24.000	782 2
Kipnis G. - Statistica in Basic	L. 16.500	210 3
Putnam B.V. - RS - 232	L. 27.500	406 8
Manuale di informatica	L. 148.000	365 7
Rougè D. - PC tools de luxe	L. 35.000	649 4
Rougè D., Hochart J.J. - Microsoft - Simulatore di volo flight simulator	L. 29.000	905 1
Siliescu D. - SoundBlaster Programmi Shareware	in preparazione	0063 5
Siliescu D. - SoundBlaster PRO. In venti lezioni	L. 19.000	971 X
Sutty G., Blair S. - Guida alle schede EGA/VGA	L. 32.000	581 1
Taboureux O. - L'informatica nello studio legale	L. 15.000	428 9
The Waite Group - Frattali per Windows - Programmi di generazione immagini	in preparazione	0042 2
Tway L. - Multimedialità I 500 Megabyte più spettacolari	in preparazione	0041 4
Vollmuyth J., Mueller T., Catarinelli A. - MIDI Computer e musica	L. 21.000	436 X
Woodward J. - La rete Novell - Guida pratica	L. 49.000	642 7
Informatica microprocessori		
Morse S. P., Isaacson E. J., Albert D. J. - Il microprocessore 80386/387	L. 43.000	403 3
Scanlon L. J. - Programmazione del microprocessore 68000	L. 20.000	219 7
Willen D.C. - Krantz J. I. - IBM PC programmazione del microprocessore 8088	L. 30.000	221 9
Informatica sistemi operativi		
Bryan M. - System 7 Macintosh	L. 29.000	771 7
Ewing K.S. - Windows	L. 48.000	538 2
Goodel T. - DR-DOS 6 - Guida pratica	L. 69.000	907 8
Minassi M. Windows 3.1	L. 32.000	976 0
Mullen R., Hoffman P., Sosinsky B. - Windows 3.1	L. 65.000	778 4
Norton P., Ashley R., Fernandez J. - DOS 5 avanzato di Peter Norton	L. 49.000	911 6
Norton P., Ashley R., Fernandez J. - DOS 6 avanzato di Peter Norton	L. 54.000	970 1
Robbins - DOS 5 - Guida completa	L. 69.000	750 4
Russel Stultz A. - DOS 5 - Autoistruzione a moduli	L. 39.000	729 6

Indice Analitico

ABS, 37
Addizione, 41
Algoritmo, 58
Ambito, 120
Annidamento, 66
Apertura, 127
Arco, 157
Argomento, 37, 112, 116
AS STRING, 45
ASC, 50
ASCII, 14
Assegnazione, 93
ATN, 37
Attributo, 163
Avvio, 5
Barra,
  dei menu, 6
  di riferimento, 6
BEEP, 177
Bidimensionale, 41
Binario, 14
Blocco, 82
Booleano, 67
BUFFER, 129
CALL, 111
Campo, 127
CDBL, 38
Cella, 25
Cerchio, 156
CGA, 147
CHDIR, 144
CHR$, 50
Ciclo, 63
CINT, 38
CIRCLE, 156
CLEAR, 21
CLNG, 38
CLOSE, 129
CLS 1, 160
CLS 2, 33
Codice, 14
  d'errore, 185
Colonna, 25
COLOR, 162
Colore, 162
Commento, 59
Componente, 91
Concatenazione, 47
Condizione, 77, 81
Confronto, 48
Congiunzione, 67
CONST, 19, 45
Contatore, 63
Controllo,
  codice di, 28
Coordinata, 147
  fisica, 161
  logica, 161
  relativa, 151
COS, 38
Costante, 19, 117
Croma, 180
CSNG, 38
CSRLIN, 31
Cursore, 31
CVD, 142
CVI, 142
CVL, 142
CVS, 142
DATE$, 54
Dati,
  aggiunta dei, 137
  formattati, 27
  immissione dei, 55, 138
  lettura dei, 100, 141
  numerici, 13
DEF, 21
DEF FN, 115
DEFSTR, 45
Dichiarazione, 20, 45

DIM, 46, 92
DIM SHARED, 120
Dimensionamento, 92, 94
Disegno, 150
Disgiunzione, 67
Distruzione, 58
Diverso, 48
Divisione, 41
DO...LOOP, 71
DRAW, 171
Durata, 177
Editazione, 113
Editor, 5
EGA, 147
Elevamento,
  a potenza, 41
Ellisse, 156
END, 59
EOF, 132
Equivalente, 67
ERASE, 93
ERR, 185
Errore, 185
Esadecimale, 143
Esclusivo, 67
Esponente, 13
Espressione, 28, 108, 117
  booleana, 67
  numerica, 41
Estrapolazione, 95
Etichetta, 58
EXIT DO, 76
EXIT FOR, 66
EXP, 38
FIELD, 140
Figura, 165
File,
  ad accesso casuale, 129
  chiusura del, 129
  creazione del, 130
  di dati, 127
  di programma, 127
  lettura del, 132,143
  scrittura del, 143
  sequenziale, 128
FILES, 144
Finestra,
  attiva, 6
  di testo, 33
  grafica, 160
  immediata, 9
  senza titolo, 8
FIX, 39
Floating Point, 13
FOR...NEXT, 63
Frequenza, 177
FUNCTION...END FUNCTION, 114
Funzione, 113
  di stringa, 50
  numerica, 37
GRAPHICS, 173
Hardware, 14
HEX$, 50
HGA, 147
IF...THEN...ELSE, 81
Immissione, 55
Implicazione, 67
INKEY$, 55
INPUT, 55
INPUT #, 132
INPUT$, 55, 133
INSTR, 50
INT, 39
Interattivo, 8
Intero, 20
Interpolazione, 95
KILL, 144
LCASE$, 51
LEFT$, 51
LEN, 52, 138
LET, 19
Lettura, 98
  binaria, 143
LINE, 151
LINE INPUT, 55
LINE INPUT #, 133
Linea,
  tratteggiata, 153
Linguaggio, 171
Lista, 85, 108
LOCATE, 31
LOG, 39
LPRINT, 32
LPRINT USING, 33
LTRIM$, 52
Lunghezza, 46, 138
Maggiore, 48
Mantissa, 13

Massima, 101
Matrice, 91
MCGA, 147
Memorizzazione, 138
Menu,
  di scelta, 77
MID$, 52
MKD$, 140
MKDIR, 144
MKI$, 140
MKL$, 140
MKS$, 140
Modulo, 41, 107
Moltiplicazione, 41
Monitor, 5
Motivo, 167
NAME, 144
Nota, 177
Notazione,
  floating point, 13
  scentifica, 13
Numerazione,
  sistema di, 14
OCT$, 50
ON ERROR GOTO, 185
ON...GOSUB, 108
OPEN, 129
Operatore, 41, 48, 67
Operazione, 104
OPTION BASE, 92
Opzione, 5
OR, 67
Ordinamento, 102
Ottava, 179
PAINT, 165
PALETTE, 164
Palette, 163
PALETTE USING, 164
Parametro, 37, 112, 116
Passaggio, 117
Passo, 63
Pausa, 122
Pixel, 147
PLAY, 178
POS, 31
Precisione, 20
PRESET, 150
Primo piano, 162
Print Using, 27
Print, 26
Procedura, 110
Programma, 58
  fine del, 59
  principale, 108
Programmazione, 1
PSET, 150
PUT #, 138
READ...DATA, 98
Record, 127
  numero del, 142
Relazione, 48
REM, 59
RESTORE, 99
RESUME NEXT, 188
Rettangolo, 153
RETURN, 108
Ricorsivo, 123
Riempimento, 165
Riferimento, 118
Riga, 9, 25
RIGHT$, 52
Risoluzione, 147
RND, 39
Rotazione, 173
Routine, 186
RSET, 46
RTRIM$, 51
RUN, 129
Salvataggio, 127
Scheda,
  grafica, 147
SCREEN, 148
Scrittura,
  binaria, 143
Segmento, 151
SELECT CASE, 84
Sfondo, 162
SGN, 40
SIN, 40
Software, 1
Sottoprogramma, 111
Sottrazione, 41
SOUND, 177
SPACE$, 30
Spazio, 30
SPC, 30
SQR, 40
Stampa,

di grafici, 173
su carta, 32
Stampante, 174
STATIC, 110, 120
STEP, 151
STILE, 153
STR$, 53
STRING$, 47
Stringa, 28, 45
SUB...END SUB, 111
Subroutine, 108
SWAP, 102
TAB, 31
Tabulazione, 31
TAN, 40
Tempo, 181
TIME$, 54
TIMER, 54
Top-down, 107
Torta, 159
Trasformazione, 15
TYPE...END TYPE, 137
UCASE$, 51
Uguale, 48
Unidimensionale, 41
VAL, 53
Valore, 93, 119
Variabile, 19
statica, 120
Versione, 140
Vettore, 91
cancellazione del, 93
VGA, 147
Videata, 6
VIEW, 160
VIEW PRINT, 33
Virgola,
mobile, 13
WHILE...WEND, 70
WIDTH, 32
WINDOW, 161
WRITE #, 130

INFORMATICA


ISBN 88-481-0019-8

Il contenuto di questo libro è di carattere:



AVANZATO

La Microsoft, società produttrice e distributrice di MS-DOS, il più diffuso sistema operativo per PC, ha dotato questo prodotto fin dalle sue prime versioni, di un linguaggio di programmazione, l'altrettanto famoso BASIC, che ha permesso a milioni di utenti non professionali di avvicinarsi per la prima volta a questa affascinante attività informatica che è la programmazione. È quindi naturale che anche l'ambiente operativo Windows, che in un certo senso raccoglie l'eredità del DOS e facilita l'uso del computer a platee più vaste di utenti, fosse dotato di una propria versione di BASIC che nel caso specifico è denominata QBASIC. Se la nuova versione del linguaggio si presenta più amichevole nella sua interazione con l'utente rispetto alle precedenti, essa presuppone tuttavia come ogni linguaggio di programmazione che l'utente abbia ben chiari i principi fondamentali di questa disciplina, prima di poter trarre vantaggio dagli aiuti in linea e dalla sintassi delle singole istruzioni richiamabili comodamente a video. Il Volume in questione che riprende e aggiorna le parti principali del precedente volume "Il Basic" dello stesso Autore, intende proprio fornire uno strumento di lavoro e una serie di suggerimenti utili a quanti intendono imparare l'affascinante tecnica della programmazione, perché vogliono assumere un controllo più completo possibile delle operazioni da far compiere al proprio computer, al di là cioè del normale utilizzo quotidiano della maggior parte degli utenti. A questa categoria di utenti il volume permetterà di passare in maniera semplice e forse - chissà - anche divertente dalla fase di utilizzatori di computer a quella di informatici.



Angelo Gallippi: *è nato a Roma nel 1946 e si è laureato in fisica nel 1971. Da allora ha alternato l'insegnamento della matematica negli istituti tecnici con la redazione di testi di informatica. Attualmente collabora, in qualità di giornalista pubblicista, con Il Sole-24 ore, Mass Media e varie riviste di Tecniche Nuove, inoltre svolge attività di consulenza e formazione informatica presso il Ministero della Pubblica Istruzione.*

L. 29.000

tecniche nuove

